GWD-E
OGF DFDL WG
dfdl-wg@ogf.org

Stephen M Hanson (IBM)
13 September 2013

Data Format Description Language (DFDL) v1.0 Experience Document 1

Status of This Document

Grid Working Document (GWD)

Copyright Notice

Abstract

This document provides experience information to the OGF community on the original Data Format Description Language (DFDL) 1.0 specification (GFD-P-R.174).

It lists and describes the non-editorial errata identified in the DFDL 1.0 specification. It contains all errata up to 2013-09-03.

All errata have been incorporated into a revised Data Format Description Language (DFDL) 1.0 specification (GFD-P-R.nnn).

Comment [SMH1]: Complete when GFD number allocated.

Contents

1. Introduction

This document has been created to list issues encountered by implementers of the original DFDL 1.0 specification [DFDL], and users of implementations of the DFDL 1.0 specification. Specifically, it records all those issues requiring a non-editorial change to the DFDL 1.0 specification, in the form of errata.

The OGF GFD process [GFD] recognises three different kinds of error that may be found in OGF specifications:

Editorial fixes. Updates to a document which are not widely announced or publicized. This category might include headers/footers, spelling, formatting, or simple wording changes for clarity.

Minor technical fixes. Updates to a document which are not simply editorial. For example, an update to an XML schema or addition to a protocol, to bring the document into agreement with current practice.

Major technical fixes. Such fixes will often require additional technical review and result in an updated or replaced document.

The following sections of this document list the errata that fall into the last two categories.

All the errata in this document have been incorporated into a revision of the DFDL 1.0 specification [DFDLREV].

2. Minor Technical Fixes

The following minor technical fixes have been identified.

2.1. *Section 7.2.2.* The ref property needs to state that circular paths are a schema definition error.

2.2. *Section 13.13.* Clarify what packed and BCD calendars mean.

- There is no need to use a separate VDP property. The only place where a decimal point can occur is for fractional seconds. This is detectable from the pattern at the boundary of 's' and 'S', ie sS.
- Property calendarPatternKind = 'explicit' must be used with binary calendar representations, as the defaults for 'implicit' use non-numeric characters. Schema definition error otherwise.
- Property binaryCalendarRep should restate the rule from property calendarPattern.
- Examples to be provided.

2.3. *Section 13.11.1.* Does not fully state the time zone symbol behaviour. It should say:

z	Time Zone: specific non-location	Text	z, zz, zzz zzzz	PDT Pacific Daylight Time
Z	Time Zone: ISO8601 basic format Time Zone: localized GMT	Text	Z, ZZ ZZZ ZZZZ	-0800, +0000 GMT-08:00, GMT+00:00
O	Time Zone: localized GMT	Text	O OOOO	GMT- GMT-08:00
v	Time Zone: generic non-location	Text	v vvvv	PT Pacific Time
V	Time Zone: short time zone ID long time zone ID exemplar city generic location.	Text	V VV VVV VVVV	uslax America/Los_Angeles Los Angeles Los Angeles Time
x	Time Zone: ISO8601 basic or extended format	Text	x xx xxx	-08, +0530, +0000 -0800, +0000 -08:00, +00:00
X	Time Zone: ISO8601 basic or extended format .The UTC indicator "Z" is used when local time offset is 0.	Text	X XX XXX	-08, +0530, Z -0800, Z -08:00, Z

Note this table reflects updates made by erratum 2.121.

2.4. *Sections 22.1.1 & 22.2.1.* Binary representations can have property lengthKind set to 'delimited'.

2.5. *Sections 22.1.2 & 22.2.2.* Complex elements can have property lengthKind set to 'endOfParent'.

2.6. *Throughout.* Specification often uses the term 'content region' but it should be more specific in terms of the grammar, and use 'SimpleContent region' or 'ComplexContent region', or both.

2.7. *Section 17.* Text says that inputValueCalc and outputValueCalc applies to simple types, which is not correct. Absorbed into erratum 3.2.

2.8. *Section 13.16.* In the description of property nilValue, state that nilLiteralCharacter test takes place on the untrimmed representation value.

2.9. *Section 12.3.3.* Clarify that when property lengthUnits is 'bytes', using property lengthKind 'implicit' for a string interprets the min/maxLength facets as byte values and not characters when parsing or unparsing.

2.10. *Section 6.3*. Bullet for logical value. State that the string must obey the lexical representation of the type.

2.11. Section 6.3. Clarify that literal white space is only ever used as list token separator, and that entities must be used if literal white space is needed as part of the property value.

2.12. *Section 6, 7.7*. Clarify that if A.xsd includes B.xsd then A can refer to a variable defined in B and reference is via QName in the usual way. This is best expressed by simply saying that DFDL QNames behave like XSDL QNames in section 6. The existing text in section 7.7 can be removed.

2.13. *Section 9.2*. Correct the grammar to reflect that a prefix length type can itself have a prefix length. This is sufficient to allow the grammar to describe the needed "one more level" of prefix (as required for modeling an ASN.1 format) without allowing recursion.

The updated grammar is in Chapter 4 of this document.

2.14. *Section 12.3.4.* Clarify that when a prefix length type itself has a prefix length, the simple types cannot be the same.

Explicitly list the property restrictions that must apply to a prefix length type to comply with modeling just SimpleContent region. It is a schema definition error if the type specifies lengthKind 'delimited' or 'endOfParent' or 'pattern' or 'explicit' where length is an expression, or a value for initiator or terminator other than empty string, or alignment other than '1', or leadingSkip or trailingSkip other than '0'.

2.15. *Section 13.6.* When property textNumberRep is 'zoned', the property description should state that base is assumed to be 10.

2.16. *Section 13.2.1*. Clarify string literal content of properties escapeCharacter, escapeEscapeCharacter, extraEscapedCharacters, escapeBlockStart and escapeBlockEnd.

- DFDL character entities are allowed
- The raw byte entity (%#r) is not allowed
- DFDL Character classes (NL, WSP, WSP+, WSP*, ES) are not allowed

2.17. *Sections 13.4, 13.6, 13.9, 13.12.* Clarify string literal content of properties textStringPadCharacter, textBooleanPadCharacter, textCalendarPadCharacter and textNumberPadCharacter.

- DFDL character entities are allowed
- The raw byte entity (%#r) is allowed subject to the restrictions already documented for these properties
- DFDL Character classes (NL, WSP, WSP+, WSP*, ES) are not allowed

2.18. *Section 13.6*. Clarify string literal content of properties textStandardDecimalSeparator, textStandardGroupingSeparator, textStandardExponentCharacter, textStandardInfinityRep, textStandardNaNRep.

- DFDL character entities are allowed
- The raw byte entity (%#r) is not allowed
- DFDL Character classes (NL, WSP, WSP+, WSP*, ES) are not allowed

2.19. *Section 13.9*. Clarify string literal content of properties textBooleanTrueRep and textBooleanFalseRep.

- DFDL character entities are allowed

- The raw byte entity (%#r) is not allowed
- DFDL Character classes (NL, WSP, WSP+, WSP*, ES) are not allowed

2.20. *Section 13.16*. Remove restriction that property nilValue only applies when representation is text. It is not clear where this originated.

Clarify string literal content of nilValue when nilKind is 'literalValue':
When representation is text:
- DFDL character entities are allowed
- The raw byte entity (%#r) is allowed
- DFDL Character classes (NL, WSP, WSP+, WSP*, ES) are allowed.

When representation is binary:
- DFDL character entities are allowed
- The raw byte entity (%#r) is allowed
- DFDL Character class ES is allowed.
- Other DFDL Character classes (NL, WSP, WSP+, WSP*) are not allowed.

Clarify string literal content of nilValue when nilKind is 'literalCharacter':
When representation is text:
- DFDL character entities are allowed
- The raw byte entity (%#r) is allowed subject to the restrictions already documented for this property
- DFDL Character classes (NL, WSP, WSP+, WSP*, ES) are not allowed.

When representation is binary:
- DFDL character entities are allowed
- The raw byte entity (%#r) is allowed.
- DFDL Character classes (NL, WSP, WSP+, WSP*, ES) are not allowed.

2.21. *Section 13.6*. Change meaning of textNumberCheckPolicy enum 'lax' to align with ICU.: "If 'lax' and dfdl:textNumberRep is 'standard' then grouping separators are ignored, leading and trailing whitespace is ignored, leading zeros are ignored, quoted characters may be omitted."

2.22. *Section 13.6*. Disallow the use of empty string for property textStandardDecimalSeparator, and state property must be set if the pattern contains a '`.`' or 'E' or '@' symbol (schema definition error otherwise).

2.23. *Section 13.6*. Allow decimal separator to be a List of DFDL String Literals or a DFDL expression. This allows modelling of the EDIFACT standard where a user can choose a dynamic decimal separator in the ISA header but '`.`' is always allowed.

2.24. *Section 13.6*. Disallow the use of empty string for property textStandardGroupingSeparator, and state property must be set if the pattern contains a '`,`' (schema definition error otherwise).

2.25. *Section 13.6*. When property textNumberPadCharacter is '0' (or an equivalent DFDL Character Entity) which it commonly is, a value of say '00000' will get trimmed to the empty string, whereas the intent is to trim to '0'. Add a new rule that says the last remaining digit is never trimmed for text numbers regardless of its value. This rule only applies to the character '0', and not to any other numeric character nor to DFDL Byte Value Entity.

2.26. *Section 13.6*. Allow the use of empty string for property textStandardExponentCharacter to model text numbers of the form nnn+mmm. Property must be set even if the pattern does not contain an 'E' symbol, to match ICU behaviour (schema definition error otherwise).

2.27. *Section 13.6.* textStandardDecimalSeparator must be ignored when the logical type is not decimal/float/double.

2.28. *Section 13.6.1.1.* Add support for ICU significant digits symbol '@'. Note that this is not needed as a change in 13.6.1.2.

2.29. *Section 13.6.1.1.* Formatting. Uses terms 'minimum/maximum integer/fraction digits' but does not define them. The term 'maximum integer digits' is defined as 309 to match the ICU default, the other terms are defined by the pattern content.

2.30. *Section 13.9*. State that textBooleanTrueRep and textBooleanFalseRep properties are used after trimming when parsing, and before padding when unparsing. If lengthKind is 'explicit' or 'implicit' and either textPadKind or textTrimKind is 'none' then the properties must have the same length else it is a schema definition error.

2.31. *Section 16.2.* State it is a processing error if the stop value is missing from the data when parsing.

2.32. *Section 12.1.1.* Clarify the note after Table 14 mean. "Specifying the implicit alignment in bits does not imply that dfdl:lengthUnits 'bits' can be specified for all simple types". It is really saying that alignmentUnits and lengthUnits are independent and have their own rules for when they are applicable.

2.33. *Section 12.3.***.** One line descriptions of 'delimited' and 'endOfParent' are not worded correctly in the property description of lengthKind, and should be improved.

2.34. *Section 12.3.2*. Rule 3 for resolving ambiguity between delimiters, which says "When the separator and terminator on a group have the same value, the separator has precedence", needs clarifying to say "When the separator and terminator on a group have the same value, then at a point where either separator or terminator could be found, the separator is tried first."

2.35. *Section 17.* InputValueCalc. Description talks about returning an empty string being ok if minLength permits this. Replace sentence with a fuller clarification that inputValueCalc value is validated like a parsed value, so schema definition error if value does not conform to base type, and validation error if validation enabled and value conforms to base type but not actual type.

2.36. *Section 16.* Spec allows occursCount to be a non-negative integer. This is superfluous as the property is only used when occursCountKind is expression. Change so that occursCount is only allowed to be a DFDL Expression.

2.37. *Section 23.3*. Clarify that DFDL expression syntax "{}" is invalid, as it results in an empty XPath 2.0 expression, which is not legal. In particular, clarify that setting a property to {} does not give the same result as setting a property to the empty string.

2.38. *Section 11*. Specification does not definitively list which binary reps are subject to byteOrder. Clarify that byteOrder applies to all Numbers and Calendars with representation binary. Specifically that is binary integers, packed decimals, BCD, binary floats, binary seconds and binary milliseconds.

2.39. *Section 13.11.1.* When parsing an xs:date or xs:datetime, if a calendarPattern doesn't specify some parts (other than time zone), say, calendarPattern="MM", then the Unix epoch 1970-01-01T00:00:00.000 is used to provide the missing parts.

Noted that if a pure month or day or year is needed, then this would be achieved by a future DFDL extension to expand the supported simple types to include xs:gMonth, xs:gDay, xs:gYear types.

2.40. *Section 13.6.1.1*. The paragraphs that describe the V symbol (virtual point) and P symbol (scaling factor) talk about 'number region'. This relates to the BNF and so it should say 'vpinteger region' instead, which is where in the pattern the V and P symbols reside. Table 20 should also be updated so that it matches the BNF.

2.41. *Section 13.6.1.2.* This section does not explicitly say that its content is effectively a delta on section 13.6.1.1. The section should be rewritten to make it clear which behaviour is the same as 13.6.1.1 and which is different.

2.42. *Section 13.6*. Clarify string literal content of property textStandardZeroRep.

- DFDL character entities are allowed
- The raw byte entity (%#r) is not allowed
- DFDL Character classes (NL, ES) are not allowed
- DFDL Character classes (WSP, WSP+, WSP*) are allowed, however, WSP* cannot appear alone as one of the string literals for this property as this would allow an empty string to match as the representation. (Consistent with not allowing the ES character class entity.)

2.43. *Section 13.11*. Property calendarTimeZone is defined as an Enum of type string, but in reality is better defined as a String constrained by a regular expression:
`(UTC)([+\-]([01]\d|\d)((([:][0-5]\d){1,2})?))?)`
See also errata 2.50 and 2.65.

2.44. *Section 13.11*. Property calendarLanguage is defined as an Enum of type string, but in reality is better defined as a String constrained by a regular expression:

`([A-Za-z]{1,8}([\-_][A-Za-z0-9]{1,8})*)`

Updated to allow underscores as well as hyphens in calendarLanguage syntax.

2.45. *Sections 13.17 and 22*. State that property useNilForDefault is only examined when xs:nillable is "true", and must be set when xs:nillable is "true".

2.46. *Section 13.6*. Allow multiple characters for property textStandardExponentCharacter to handle representations like `1.23`**`x10^`**`4` as ICU allows that. Note that property name will therefore change to textStandardExponentRep.

2.47. *Section 13.6.* Change name of property textStandardNanRep to textStandardNa**N**Rep to reflect common usage of NaN and avoid typographical errors in models.

2.48. *Section 14.1*. Spec states that an empty sequence that is the content of a complex type is a schema definition error. Many schema processors are not able to distinguish this condition from a complex type with no content at all (it is not required to do so by the XML Schema specification). As a complex type with no content is not useful in DFDL, change the spec to state that both conditions are schema definition errors.

2.49. *Section 23.3.* Clarify that when a property can be either a DFDL String Literal or a DFDL Expression, then if the value is a DFDL String Literal and the first character is '{' then it MUST be escaped as '{{'. For such a property, a value '{xxx' will be treated as an (invalid) expression, and not as a string literal.

2.50. *Section 13.11*. The calendarTimeZone property is used to supply a time zone when there is none in the data (and by implication none in the pattern). However this means DFDL is not compatible with XML Schema 1.0 where "no time zone" is an allowable state for a calendar infoset value. Further, XML Schema 1.0 validation validates a calendar value against facets according to rules that cater for "no time zone" [XSDL2]. It is desirable therefore for DFDL to permit a calendar value to have "no time zone". Accordingly, the calendarTimeZone property will allow a value of empty string to indicate "no time zone".

2.51. *Section 13.11.* The calendarTimeZone property will apply when parsing only. This avoids any problem with values changing after validation has taken place.

2.52. *Section 13.11.1.* DFDL only allows 'y' as year symbol in dfdl:calendarPattern. This only allows positive values, any negative value is ignored unless the 'G' (era) symbol is also specified, and there is no year 0 (which means that negative astronomical dates are one year out). DFDL will also support the ICU 'u' extended year symbol which allows year 0 (means 1BC) and corresponds to astronomical years.

2.53. *Section 13.13.* Property binaryCalendarEpoch is used when the binaryCalendarRep is either binarySeconds or binaryMilliseconds, and is of type xs:dateTime. It is allowable to omit the time zone component from the binaryCalendarEpoch property value, and if this occurs UTC is used as the time zone.

2.54. *Section 13.11.* ICU has some lax behaviour when parsing numbers using the supplied textNumberPattern, using property textNumberCheckPolicy. Erratum 2.21 corrects the definition of behaviour when 'lax' is specified but does not state what the base behaviour is when 'strict' is specified. This should be stated as follows:

"If 'strict' and dfdl:textNumberRep is 'standard' then the data must follow the pattern with the exceptions that digits 0-9, decimal separator and exponent separator are always recognised and parsed."

2.55. *Section 9.2*. Grammar terminal FinalUnusedRegion is intended to handle unmodeled bytes in the data that arise due to 'specified length' settings of lengthKind on a complex xs:element and choiceLengthKind on a xs:choice. It is not doing so correctly when a terminator is present on a xs:sequence or xs:choice. To fix this, the terminal is removed from the grammar and replaced by two new terminals ElementUnused and ChoiceUnused.

The updated grammar is in Chapter 4 of this document.

2.56. *Section 13.13*. State that when property binaryCalendarRep is set to 'binaryMilliseconds' or 'binarySeconds', it is a schema definition error if the type is xs:time or xs:date. This is because when unparsing it is not possible to obtain a milliseconds or seconds value from just an xs:time or xs:date and the epoch.

2.57. *Section 13.13*. State that when property binaryCalendarRep is set to 'binarySeconds' or 'binaryMilliseconds', the value in the data is treated as signed. This lets DFDL support POSIX/Unix times, which are allowed to be negative.

2.58. *Section 13.16*. State that property nilValueDelimiterPolicy is ignored when property nilKind is set to 'logicalValue', and the behaviour of the DFDL processor is to expect delimiters when parsing and to output delimiters when unparsing, if delimiters are specified for the element. This is to simplify implementations.

2.59. *Section 7.1.3.3*. State that short form property syntax is not allowed on the xs:schema object as an equivalent to the element form property syntax of the default dfdl:format (or any other global DFDL) annotation.

2.60. *Section 13.3*. Remove redundancy from the names of some of the bidirectional text properties, specifically textBidiTextOrdering becomes textBidiOrdering, and textBidiTextShaped becomes textBidiShaped.

2.61. *Section 9.2*. The child content of xs:sequence and xs:choice are almost the same, so the grammar can be refactored to remove duplication.

The updated grammar is in Chapter 4 of this document.

2.62. *Section 11.* Clarify that when the encoding property is specified as 'UTF-8' then that is a strict definition of UTF-8 and does not include variants such as CESU-8. This is in keeping with ICU's interpretation of UTF-8.

2.63. *Sections 12.2, 14.2*. Properties that use the empty string as a special value to switch off use of the property, and that allow the value to be a DFDL expression, should not be able to set empty string by evaluating the expression. It must be possible to evaluate statically whether the property is used or not. This affects properties initiator, terminator, separator. It is a schema definition error if an expression returns the empty string.

2.64. *Section 6.3.1*. Update to say that empty string is not allowed as a string literal value, unless explicitly stated otherwise in the description of a property, in which case any semantic must be a special behaviour of the property and not the literal empty string (for which DFDL provides entity %ES;).

2.65. *Section 13.11.* Property calendarTimezone is changed to accept either a UTC offset or an Olson format time zone. Property calendarObserveDST is changed so that it is only used when calendarTimeZone is Olson format. If it is a UTC offset then calendarObserveDST is ignored.

2.66. *Section 13.11*. When unparsing and property calendarPattern contains a formatting symbol for time zone (zzz, Z, VVVV etc) and the infoset value does not contain a time zone, it is a processing error. This matches the behaviour on parsing when the data does not contain a time zone but the pattern does.

2.67. *Section 4.1.2*. The second paragraph of the description of [dataValue] should be replaced with:

"For information items of datatype xs:string, the value is an ordered collection of unsigned 16-bit integer codepoints each having any value from 0x0000 to 0xFFFF. Where defined, these are interpreted as the ISO646 character codes. Codepoints disallowed by ISO 10646, such as 0xD800 to 0xDFFF are explicitly allowed by the DFDL infoset. The codepoints of the string are stored in 'implicit' (also known as logical), left-to-right bidirectional ordering and orientation. DFDL's infoset represents Unicode characters with character codes beyond 0xFFFF by way of surrogate pairs (2 adjacent codepoints) in a manner consistent with the UTF-16 encoding of ISO 10646."

2.68. *Section 13.11.1*. Calendar formatting symbols 'I' and 'T' are intended as a short-hand way of accepting a subset of ISO 8601 variants. However the description of the behaviour is not correct and is unnecessarily complex. The 'T' symbol is dropped altogether, and the 'I' symbol behaviour is defined as the following:

"The 'I' symbol must not be used with any other symbol with the exception of 'escape for text'. It represents calendar formats that match those defined in the restricted profile of the ISO 8601 standard proposed by the W3C at http://www.w3.org/TR/NOTE-datetime. The formats are referred to as 'granularities'.

- xs:dateTime. When parsing, the data must match one of the granularities. When unparsing, the fullest granularity is used.
- xs:date. When parsing, the data must match one of the date-only granularities. When unparsing, the fullest date-only granularity is used. 'IU' is permitted for xs:date but the 'U' is ignored as there is no time zone in the date-only granularities.
- xs:time. When parsing, the data must match only the time components of one of the granularities that contains time components. When unparsing, the time components of the fullest granularity are used. The literal 'T' character is not expected in the data when parsing and is not output when unparsing.
- The number of fractional second digits supported is implementation dependent but must be at least one.

- For a granularity that omits components, when parsing the values for the omitted components are supplied from the Unix epoch 1970-01-01T00:00:00.000."

2.69. *Section 23.3*. The last paragraph is inconsistent with the rest of the section. It should say:

"The result of evaluating the expression must be a single atomic value of the type expected by the context, and it is a schema definition error otherwise. Some XPath expressions naturally return a sequence of values, and in this case it is also schema definition error if an expression returns a sequence containing more than one item."

The sentence "If the expression returns an empty sequence it will be treated as returning nil" is removed.

2.70. *Section 12.2, 14.2.* Clarify the parser matching algorithm used for properties initiator, terminator and separator.

When parsing, the list of values is processed in a greedy manner, meaning it takes all the initiators, that is, each of the string literals in the white space separated list, and matches them each against the data. In each case the longest possible match is found. The initiator with the longest match is the one that is selected as having been 'found', with length-ties being resolved so that the matching initiator is selected that is first in the order written in the schema. Once a matching initiator is found, no other matches will be subsequently attempted (ie, there is no backtracking).

2.71. *Section 13.16*. Clarify that nilValue is sensitive to ignoreCase when nilKind is 'literalValue' or 'logicalValue', to be consistent with properties such as textBooleanTrueRep, but not when nilKind is 'literalCharacter', to be consistent with properties such as textBooleanPadCharacter.

2.72. *Section 12.3.6*. Additional constraints and clarifications apply to the use of lengthKind 'endOfParent' beyond those already documented:

The parent element lengthKind must not be 'implicit' or 'delimited'.

When looking for end of parent, the parser is not sensitive to any in-scope terminating delimiters.

If the element is in a sequence then:
- the sequence must be the content of a complex type
- the separatorPosition of the sequence must not be 'postFix'
- the sequenceKind of the sequence must be 'ordered'
- no terminator on the sequence
- no trailingSkip on the sequence
- no floating elements in the sequence

If the element is in a choice where choiceLengthKind is 'implicit' then
- the choice must be the content of a complex type
- no terminator on the choice
- no trailingSkip on the choice

A simple element must have either type xs:string or representation 'text' or type xs:hexBinary or (representation 'binary' and binaryNumber/CalendarRep 'packed', 'bcd, 'ibm4690Packed').

As noted in erratum 2.5, a complex element can have 'endOfParent'. If so then its last child element can be any lengthKind including 'endOfParent'.

An element with ‘endOfParent’ must be the last thing in its ‘box’. A ‘box’ is defined as a portion of the data stream that has an established length prior to the parsing of its children. Specifically a box is either:

- A complex element with lengthKind 'explicit', 'prefixed' or 'pattern' AND no (sequence right framing or sequence postfix separator or choice right framing)
- A choice with choiceLengthKind 'explicit'

When unparsing, if the parent is a complex element with lengthKind ‘explicit’ or a choice with choiceLengthKind ‘explicit‘ then the element with lengthKind ‘endOfParent’ is padded or filled in the usual manner to the required length.

2.73. *Section 12.3.6*. An element with lengthKind ‘endOfParent’ is allowed to be the root element of a parse or unparse.

2.74. *Sections 13.2.1, 22.2.1*. During unparsing, the application of escape scheme processing should take place before the application of the emptyValueDelimiterPolicy property.

2.75. *Section 4.1.1*. Replace the existing description of the Document Information Item’s [schema] member with ‘This member is reserved for future use’.

2.76. *Section 12.3.4*. When property prefixIncludesPrefixLength is ‘yes’ there are some restrictions that need to be added to enable reliable lengths to be calculated:

- If the prefix type is lengthKind 'implicit' or 'explicit' then the lengthUnits properties of both the prefix type and the element must be the same.

2.77. *Sections 12.3.4, 12.3.2*. The sections for lengthKind 'prefixed' and 'delimited' need the equivalent of Table 16 to express their rules for binary data.

2.78. *Section 12.3.4*. Add a note to cover the scenario where lengthUnits is 'bits' and lengthKind is 'prefixed'. When parsing, any number of bits can be precisely extracted from the data stream, but when unparsing the number of bits written will always be a multiple of 8 as the Infoset does not contain bit-level information.

2.79. *Section 13.6.1.1*. Clarify text number pattern rules for use of V and P symbols in conjunction with # symbol.

- A pattern with a V symbol must not have # symbols to the right of the V symbol.
- A pattern with P symbols at the left end must not have # symbols .
- A pattern with P symbols at the right end can have # symbols.

2.80. *Section 13.6.1.1*. Clarify text number pattern rules for use of V and P symbols in conjunction with @ and E and * symbols.

- A pattern with a V symbol must not have @ or * symbols.
- A pattern with P symbols must not have @ or E or * symbols.

This means that a V symbol and an E symbol may occur in the same text number pattern. The BNF in Figure 5 is revised to allow this.

2.81. *Section 15.2*. The specification originally says “On unparsing the choice branch supplied in the infoset is output”. This does not handle the case where one or more branches of a choice is a sequence or a choice (or a group ref to such). Here, the element in the Infoset is one of the children of the branch sequence but it might not be the first in the sequence, or the element in the Infoset is one of the children of the branch choice. To handle this scenario, the element in the Infoset is used to search the choice branches in the schema, in schema definition order, but without looking inside any complex elements. If the element occurs in a branch then that branch is chosen. If the chosen branch causes a processing error, no other branches are chosen (that is, there is no backtracking).

To avoid any unintended behaviour, a branch sequence may be wrapped in an element.

2.82. *Section 12.3.5*. The behaviour for unparsing when lengthKind is 'pattern' is the same as for 'delimited', ie, for a simple element use textPadKind to determine whether to pad, for a complex element the length is that of the ComplexContent region.

Table 16 can accordingly be deleted.

2.83. *Section 23.3*. Clarifications on what is returned by an expression.

- Every property that accepts an expression must state exactly what the expression is expected to return
- To ensure the returned value is of the correct type, use XPath constructors or the correct literal values
- What is returned lexically by an expression follows XPath 2.0 rules, which this is not the same as xs:default and xs:fixed lexical content.
- No extra auto-casting is performed over and above that provided by XPath 2.0. XPath 2.0 has rules for when it promotes types and when it allows types to be substituted. These are in Appendix B.1 of the XPath 2.0 spec [XPATH2].
- If the property is not expecting an expression to return a DFDL string literal, the returned value is never treated as a DFDL string literal.
- If expecting expression to return a DFDL string literal, the returned value is always treated as a DFDL string literal.
- Within an expression, a string is never interpreted as a DFDL string literal

2.84. *Section 23.5.3*. The dfdl:property() function is removed.

2.85. *Section 23.5.3*. Three new functions are provided to assist in the creation of expressions that return and manipulate DFDL string literals.

dfdl:encodeDFDLEntities ($arg)	Returns a string containing a DFDL string literal constructed from the $arg string argument. If $arg contains any '%' and/or space characters, then the return value replaces each '%' with '%%' and each space with '%SP;', otherwise $arg is returned unchanged. Use this function when the value of a DFDL property is obtained from the data stream using an expression, and the type of the property is DFDL String Literal or List of DFDL String Literals, and the values extracted from the data stream could contain '%' or space characters. If the data already contains DFDL entities, this function should not be used.
dfdl:decodeDFDLEntities ($arg)	Returns a string constructed from the $arg string argument. If $arg contains syntax matching DFDL Character Entities syntax, then the corresponding characters are used in the result. Any characters in $arg not matching the DFDL Character Entities syntax remain unchanged in the result. It is a schema definition error if $arg contains syntax matching DFDL Byte Value Entities syntax. Use this function when you need to create a value which contains characters for which DFDL Character Entities are needed. An example is to create data containing the NUL (character code 0) codepoint. This character code is not allowed

	in XML documents, including DFDL Schemas; hence, it must be specified using a DFDL Character Entity. Within a DFDL Expression, use this function to obtain a string containing this character.
dfdl:containsDFDLEntities ($arg)	Returns a Boolean indicating whether the $arg string argument contains one or more DFDL entities.

2.86. *Section 24.* State that DFDL regular expressions do not interpret DFDL entities.

2.87. *Section 12.3.7*. State that when unparsing a specified length element of type xs:hexBinary, and the simple content region is larger than the length of the element in the Infoset, then the remaining bytes are filled using the fillByte property. (The fillByte is *not* used to trim an element of type xs:hexBinary when parsing.)

2.88. *Section 13.5*. Add support for HP NonStop Tandem zoned decimals. In this architecture, the negative sign is incorporated in the last byte of the number in the usual manner, but the overpunching occurs on the highest bit (ie, value 8) of the byte. Consequently, a new enum value 'asciiTandemModified' is added to property textZonedSignStyle.

Because the overpunching is on the highest bit, it means the resultant bytes are not code points in standard ASCII, so the modeller must specify an encoding like ISO-8859-1 in order for such zoned decimals to parse without an encoding error.

2.89. *Section 12.1*. In the description of the alignment property, remove the rule that states 'The alignment of a child component must be less than or equal to the alignment of the parent element, sequence or choice'. It is overly restrictive.

2.90. *Sections 12.3, 12.3.7.2*. Additionally allow lengthUnits 'bits' to apply to binary signed integer types, to support the modeling of signed integer bit fields in the C language. The physical bits are interpreted as a two's complement integer. However it is a schema definition error for a signed integer type if the length is 1 bit.

2.91. *Section 12.3.4*. State that the global simple type referenced by prefixLengthType only obtains values for missing properties from its own schema's default dfdl:format annotation. If the using element resides in a separate schema, the simple type does not pick up values from the element's schema's default dfdl:format annotation.

2.92. *Section 13.6.* When property textNumberRep is 'zoned', the property description should state that 'zoned' is only allowed for EBCDIC encodings or ASCII compatible encodings (schema definition error otherwise).

2.93. *Sections 13.6, 13.7*. State that when unparsing a number and excess precision is supplied in the Infoset and rounding is not in effect, it is a processing error. Applies to text numbers when rounding is not enabled (matches ICU behaviour), and to binary numbers (always no rounding).

2.94. *Sections 6.3.1.3, 12.2*. Correct the wording for NL mnemonic in Table 5 to make it clear that when parsing it means either `%LF;` or `%CR;` or `%CR;%LF%` or `%NEL;` or `%LS;` and not combinations of those. Similarly, state that outputNewLine can only be either `%LF;` or `%CR;` or `%CR;%LF%` or `%NEL;` or `%LS;` and not combinations of those.

2.95. *Section 12.1*. State that if representation is text or type is string, then alignment is determined by character set encoding. Most encodings are 8-bit (including those with 16-bit codepoint size like UTF-16).

Some implementations may include encodings which are not 8-bit aligned. The encoding US-ASCII-7bit-packed is 1-bit aligned. A character code occupies only 7 bits in this encoding, so character codes can begin on any bit boundary.

See also erratum 2.107 which adds the US-ASCII-7bit-packed encoding.

Section 12.1.1 is amended.

The table of explicit alignments, table 14, is modified. The column for Text is changed. The value 8, which appears in all entries in this column is replaced by "encoding dependent"

A new section is added: **Mandatory Alignment for Textual Data**.

We use the term textual data to describe data with dfdl:representation="text", as well as data being matched to delimiters (parsing) or output as delimiters (unparsing), and data being matched to regular expressions (parsing only - as in a dfdl:assert with testKind='pattern').

Textual data has mandatory alignment that is character-set-encoding dependent. That is, these mandates come from the character set specified by the dfdl:encoding property.

When processing textual data, it is a schema definition error if the dfdl:alignment and dfdl:alignmentUnits properties are used to specify alignment that is not a multiple of the encoding-required mandatory alignment.

If the data is not aligned to the proper boundary for the encoding when textual data is processed, then bits are skipped (parsing) or filled from dfdl:fillByte (unparsing) to achieve the mandatory alignment.
All character set encodings except those listed specifically below or specified by a particular DFDL implementation have mandatory alignment of 8-bit/1-byte.

- US-ASCII-7bit-packed, the alignment is 1-bit (textual data in this encoding may appear on any bit boundary, i.e., no byte alignment is required).

2.96. *Section 23.5.3*. Changes to the DFDL-specific functions for use with arrays.

The following function is renamed:
- dfdl:position() -> dfdl:occursIndex()

The function may be used on non-array elements.

The following functions are removed:
- dfdl:count()
- dfdl:countWithDefault()

Their use is replaced by standard XPath 2.0 function fn:count().

2.97. *Section 12.3.2*. Additionally allow lengthKind 'delimited' for elements of simple type xs:hexBinary.

2.98. *Section 13.7*. State that the maximum allowed value for two's complement binary integers is implementation independent but must be at least 8 bytes.

2.99. *Section 3, 13.7, 13.13 and others*. Add support for the IBM 4690 point of sale variant of a packed decimal. This has the following characteristics:

- Nibbles represent digits 0 - 9 in the usual BCD manner
- A positive value is simply indicated by digits
- A negative number is indicated by digits with the leftmost nibble being xD
- If a positive or negative value packs to an odd number of nibbles, an extra xF nibble is added on the left

Existing properties binaryNumberRep and binaryCalendarRep each take a new enum 'ibm4690Packed'. For numbers, properties byteOrder and binaryDecimalVirtualPoint actively apply. For calendars, properties byteOrder calendarPatternKind and calendarPattern actively apply (same restrictions as for 'packed' and 'bcd'). Property 'binaryPackedSignCodes' does not apply. Property 'binaryNumberCheckPolicy' applies but has no effect.

Where the DFDL specification provides for general behaviours for 'packed' and 'bcd', those behaviours apply also to 'ibm4690Packed'. Specifically:

- The same lengthKind enums and rules apply.
- There is no rounding when unparsing, so a value that can't be accommodated is a processing error.
- If logical type is unsigned and a negative value is received, it is a processing error.
- If invalid bytes are parsed, it is a processing error.

For ease of adding this erratum, a new Glossary definition is added to define a generic 'packed decimal' and this term should be used as appropriate throughout the specification.

2.100. *Section 12.3.1*. State that when unparsing an element with lengthKind 'explicit' and where length is an expression, then the data in the Infoset is treated as variable length and not fixed length. The behaviour is the same as lengthKind 'prefixed'.

2.101. *Section 23.4.* The BNFL for DFDL expressions allows a variable to appear as a path segment. This is not supported by DFDL, which only allows variables to return a simple value, and XPath does not permit variables to return simple values in path segments.

2.102. *Section 23*. State that it is a schema definition error if an array element appears as a segment in a path location and is not qualified by a predicate.

2.103. *Section 12.1*. Clarify that when the alignment properties are applied to an array element, the properties are applied to each occurrence of the element (as implied by the grammar).

2.104. *Section 13.11.1*. State that when parsing a calendar element with binaryCalendarRep 'packed', 'bcd' or 'ibm4690Packed' then the nibbles from the data are converted to text digits without any trimming of leading or trailing zeros, and the result is then matched against the calendarPattern according to the usual ICU rules.

2.105. *Section 9.1.1*. State that the presence of a separator is not sufficient to cause the parser to assert that a component is known to exist.

2.106. *Section 13.6*. State that textStandardDecimalSeparator, textStandardGroupingSeparator, textStandardExponentRep, textStandardInfinityRep, textStandardNanRep and textStandardZeroRep must all be entirely distinct from one another, and it is a schema definition otherwise. This is in the interests of clarity, and is an extra constraint compared to ICU. If any property value is an expression, the checking of this constraint cannot take place until processing.

2.107. *Section 11*. The list of enums for the encoding property is extended to include 'US-ASCII-7-bit-packed' in order to support data formats where ASCII characters are encoded in 7 bits with no padding bit. Note that the new enum is neither a CCSID or an IANA charset.

The encoding 'US-ASCII-7-bit-packed' is 1-bit aligned.

The new enum is not in the set of encodings that a DFDL processor must accept in order to be minimally conformant.

2.108. *Section 3.* Update the Glossary concerning annotations, as follows, and use the new or changed terms as appropriate throughout the specification:

- *Add:* Annotation point - A location within a DFDL schema where DFDL annotation elements are allowed to appear.
- *Add:* Statement annotations - The annotation elements dfdl:assert, dfdl:discriminator, dfdl:setVariable, and dfdl:newVariableInstance. Also called DFDL Statements.
- *Add:* Defining annotations - The annotation elements dfdl:defineFormat, dfdl:defineVariable, and dfdl:defineEscapeScheme
- *Change:* Format annotations - The annotation elements dfdl:format, dfdl:element, dfdl:simpleType, dfdl:group, dfdl:sequence, and dfdl:choice.
- *Change*: Physical Layer - A DFDL Schema adds DFDL annotations onto an XSDL language schema. The annotations describe the physical representation or physical layer of the data.
- *Add:* Resolved set of annotations - When DFDL annotations appear on a group reference and the sequence or choice of the referenced global group, or appear among an element reference, an element declaration, and its type definition, then they are combined together and the resulting set of annotations is referred to as the *resolved set of annotations* for the schema component.

2.109. *Section 6.2***.** Clarify that at any single annotation point of the schema, there can be only one format annotation (as defined in 2.108).

2.110. *Section 7.3.1, 7.4.1.* When testKind is 'pattern' for an assert or discriminator:

- The pattern is applied to the data position corresponding to the beginning of the representation. Consequently the framing (including any initiator) is visible to the pattern.
- It is a schema definition error if there is no value for encoding in scope.
- It is a schema definition error if alignment is other than 1.
- It is a schema definition error if leadingSkip is other than 0.

2.111. *Sections 5.2, 23.5.3*. Correct the XML Schema facets and attributes that are used by the dfdl:checkConstraints() function. Specifically, the function does not use the default, minOccurs and maxOccurs attributes.

2.112. *Section 3.* Update the Glossary concerning arrays, as follows, and use the new or changed terms as appropriate throughout the specification:

- *Remove:* Scalar Element
- *Remove:* Fixed-Occurrence Item
- *Remove:* Variable Occurrence Item
- *Remove:* Optional Item
- *Remove:* Number Of Occurrences
- *Change:* Required Element. An element declaration or reference where minOccurs is greater than zero.
- *Change:* Optional Element. An element declaration or reference where minOccurs is equal to zero.
- *Add:* Fixed Array Element. An array element where minOccurs is equal to maxOccurs.
- *Add:* Variable Array Element. An array element where minOccurs is not equal to maxOccurs.
- *Add:* Occurrence. An instance of an element in the data, or an item in the DFDL Infoset.
- *Add:* Count. The number of occurrences of an element. .
- *Add:* Index. The position of an occurrence in a count, starting at 1.
- *Add:* Required Occurrence. An occurrence with an index less than or equal to minOccurs.
- *Add:* Optional Occurrence. An occurrence with an index greater than minOccurs.

2.113. *Section 23*. Clarify that because of functions like fn:count(), the DFDL restriction on XPath sequences with length > 1 in reality applies to what a DFDL expression returns, and not what happens internally within an expression during evaluation.

DFDL implementations may use off-the-shelf XPath 2.0 processors, but will need to pre-process DFDL expressions to ensure that the behaviour matches the DFDL specification:

1. Ensure that what is returned as the result is not a sequence with length > 1 by appropriate use of fn:exactly-one()
2. Check for the disallowed use of those XPath 2.0 functions that are not in the DFDL subset

This requires that fn:exactly-one() is added to the list of supported XPath functions.

2.114. *Section 23*. DFDL implementations MUST comply with the error code behaviour in Appendix G of the XPath 2.0 spec [XPATH2] and map these to the correct DFDL failure type. All but one of XPath's errors map to a schema definition error. The exception is XPTY0004, which is used both for static and dynamic cases of type mismatch. A static type mismatch maps to a schema definition error, whereas a dynamic type mismatch maps to a processing error. A DFDL implementation should distinguish the two kinds of XPTY0004 error if it is able to do so, but if unable it should map all XPTY0004 errors to a schema definition error.

2.115. *Section 13.15*. Situations can arise where taking an Infoset, unparsing it, and reparsing it will result in a second Infoset that is not the same as the original. Specifically, this may occur when empty strings or values that map to nil values appear in the Infoset. This information needs adding.

This is covered in a separate DFDL experience document [DFDLX2].

2.116. *Sections 7.7, 7.8, 7.9*. To set empty string as the default value of a defineVariable or newVariableInstance annotation requires that the defaultValue attribute is used or an expression {“”} must be used as the element value. Similarly for setting empty string as the value of a setVariable annotation; use the value attribute or an expression as element value.

2.117. *Section 13.2.1*. Clarify that a padding character is not escaped by an escape character. When parsing, padding characters are trimmed without reference to an escape scheme. When unparsing, padding characters are added without reference to an escape scheme.

2.118. *Sections 11, 12.3.7.1.1*. The encoding UCS-2 is not in the list of IANA encodings nor is it a CCSID. Its use in the DFDL specifications should be removed.

2.119. *Sections 3, 12.3.5.* Update the definitions of ‘Delimiter scanning’, and ‘Scan’

- Delimiter scanning - When parsing, the process of scanning for a specific item in the input data which marks the end of an item, or the beginning of a subsequent item is referred to as delimiter scanning. Delimiter scanning also takes into account escape schemes so as to allow the delimiters to appear within data if properly escaped.
- Scan – Examine the input data looking for delimiters such as separators and terminators, or matches to regular expressions.

The term scannable alone is not in the glossary, as its meaning is implied by the definition of scan.

See also erratum 3.9.

2.120. *Sections 2.2 and 2.3*. Clarify which errors are schema definition errors and which are processing errors.

The following are processing errors:

- Arithmetic Errors
    - Division by zero

- Integer Arithmetic Underflow
- Integer Arithmetic Overflow
  - Note: Floating point math can produce NaN (Not a Number) values. This is not an error, nor are properly typed operations on floating point NaN values.

- Expression Errors
  - Dynamic Type Error – unable to convert to target type
    - Example: non-digits found in string argument to xs:int(…) constructor.
    - Note: if a DFDL Implementation cannot distinguish Dynamic Type Errors from Static Type Errors, then a Dynamic Type Error should cause a Schema Definition Error
  - Index out of bounds error – index not <= number of occurrences, or is < 1.
    - Note: same error for dfdl:testBit if bitPos is not 1..8, or for character positions in a string-value
  - Indexing of non-array non-optional element
    - Example: x[1] when x is declared and has both minOccurs="1" and maxOccurs="1" explicitly, or by not stating either or both of them.
  - Illegal argument value (correct type, illegal value)
- Parse Errors
  - Delimiter not found
  - Data not convertible to type
  - Assertion failed
  - Discriminator failed
  - Required occurrence not found
  - No choice alternative successfully parsed.
  - Character set decoding failure and dfdl:encodingErrorPolicy='error'
- Unparsing Errors
  - Truncation scenarios where truncation is being disallowed
  - Rounding error – rounding needed but not allowed. (Unparsing)
  - No choice alternative successfully unparsed.
  - Character set encoding failure and dfdl:encodingErrorPolicy='error'
- Implementation Limit Errors - Implementations can have fixed or adjustable limits that some formats and some data may exceed at processing time. This specification does not further specify what these errors are, but some possible examples are:
  - Data longer than allowed for representation of a given data type
    - Example: exceed maximum length of representation of xs:decimal in dfdl:representation="text".
  - Expression references too far back into infoset (parsing)
  - Expression references too far forward into infoset (unparsing)
  - Number of array elements exceeds limit.
- Regular expression exceeds time limit

The following are schema definition errors, regardless of whether they are detected in advance of processing or once processing begins:

- Errors in XML Schema Construction and Structure
  - See XML Schema Specification Section 5.1
- Use of XSD constructs outside of DFDL subset
- Implementation Limitations
  - Use of DFDL schema constructs not supported by this implementation.

            - Example: xs:choice is an optional part of the DFDL specification (see section 21). If not supported, it must be rejected as a Schema Definition Error.
            - Example: use of packed-decimal when it is not supported by the implementation.
            - Example: use of dfdl:assert when it is not supported by the implementation (See Spec section 21 on DFDL Subsets)
            - Note: Unrecognized DFDL properties or property values can produce a Schema Definition Warning and an implementation can attempt to process data despite the warning.
        - Exceeding limits of the implementation for schema size/complexity
            - Example: schema too large – simply a limit on how large the schema can be, how many files, how many top-level constructs, etc.
- Schema Not Valid
    - See XML Schema Specification Section 5.2
- UPA violation (Unique Particle Attribution)
- Reference to DFDL global definition not found
    - Format definition (dfdl:defineFormat)
    - Escape schema definition (dfdl:defineEscapeScheme)
    - Variable Definition (dfdl:defineVariable)
- DFDL Annotations not well-formed or not valid
- DFDL Annotations Incompatible
    - E.g., dfdl:assert and dfdl:discriminator at same combined annotation point, or more than one format annotation at an annotation point.
- DFDL Properties and their values
    - Property not applicable to DFDL annotation
    - Property value not suitable for property
    - Property conflict
        - Between Element Reference and Element Declaration
        - Between Element Declaration and Simple Type Definition
        - Between Simple Type Definition and Base Simple Type Definition
        - Between Group Reference and Sequence/Choice of Group Definition
    - Required property not found
- Expressions
    - Expression syntax error
    - Named child element doesn't exist – E.g., /a/b, and there is no child b in existence.
        - Note: no child *possible* in the schema is a different error, but also a Schema Definition Error, as /a/b would not have a type in that case.
        - Note: This is an SDE, as schema authors are advised to use fn:exists(…) to test for existence of elements when it is possible that they not exist.
    - Variable read but not defined
    - Variable assigned after read
    - Variable assigned more than once
    - Static Type error – type is incorrect for usage
        - Note: if an implementation is unable to distinguish Static Type Errors from Dynamic Type Errors, then both should cause Schema Definition Errors.
    - Path step definition not found – e.g., /a/n:b but no definition for n:b as local or global element.

- Not enough arguments for function
- Expression value is not single node
  - Most DFDL expression contexts require an expression to identify a single node, not an array (aka sequence of nodes). There are a few exceptions such as the fn:count(…) function, where the path expression must be to an array or optional element.
- Expression value is not array element or optional element.
  - Some DFDL expression contexts require an array or an optional element.
  - Example: The fn:count(...) function argument must be to an array or optional element. It is an SDE if the argument expression is otherwise.

- Regular Expressions
  - Syntax error

2.121. *Section 13.11.1.* To match revised behaviour from ICU 51, the following changes are made to the DFDL calendar pattern symbols:

- Drop the DFDL-specific 'U' symbol
- Add support for new 'x' and 'X' symbols (x, xx, xxx, X, XX, XXX only)
- Add support for all variations of the new 'O' symbol
- Adopt revised semantic for all variations of the 'V' symbol
- Adopt revised semantic for all variations of the 'z' symbol.
- Adopt revised semantic for all variations of the 'Z' symbol (but ZZZZZ not supported).

Reference is the ICU SimpleDateTime class at http://icu-project.org/apiref/icu4j/com/ibm/icu/text/SimpleDateFormat.html. Erratum 2.3 updated.

2.122. *Section 5.1.* Allow explicit setting of minOccurs = '1' and/or maxOccurs = '1' on model groups, as this is the equivalent to omitting the properties.

2.123. *Throughout*. Do not use the 'xs' prefix for XSD attributes as it is not strictly correct. Instead use the phrase 'XSD xxx property'.

2.124. *Section 23.5.3.* State that it is a schema definition error if the $node argument of dfdl:checkConstraints( ) function is a complex element.

2.125. *Section 12.3.5.* "The DFDL processor scans the data stream to determine a string value that is the longest match to a regular expression." The pattern itself dictates greediness so the word 'longest' is not needed and is removed.

2.126. *Section 3.* Correct the current inconsistencies when referring to different kinds of DFDL property. Use the revised terms as appropriate throughout the specification:

- *Change.* Format property – a DFDL property carried on a DFDL format annotation.

- *Change.* Representation property – a format property that is used to describe a physical characteristic of a component. Such a property will apply to one or more grammar regions of the component.

- *Add:* Non-representation property – a format property that is not a representation property, specifically dfdl:ref, dfdl:hiddenGroupRef, dfdl:inputValueCalc, dfdl:outputValueCalc, dfdl:choiceBranchKey, dfdl:choiceDispatchKey.

Note that ‘property’ should be used instead of ‘attribute’ for all properties that are carried on any DFDL annotation, even when an XML attribute is the only way that a property may be specified. This is consistent with XML Schema where ‘attribute’ is technically just a rendering of a property.

Note that dfdl:escapeSchemeRef is considered to be a representation property.

2.127. *Section 13.11.* The calendar pattern symbols Z, ZZ and ZZZ are equivalent. ICU prefers that Z is used singly, so the calendar pattern used for an xs:time object when calendarPatternKind is ‘implicit’ is changed to ‘HH:mm:ssZ’.

2.128. *Section 17.* State that when an element which carries the inputValueCalc property appears in a sequence that has a separator, no separator is associated with the element. When parsing, no separator is expected in the input data. When unparsing, no separator is written to the output data.

2.129. *Section 15.* A choice that declares no branches in the DFDL schema is a schema definition error. This interpretation is consistent with the rule that says each declared branch must have minOccurs > 0.

2.130. *Section 13.2.* In the description of textOutputMinLength, delete the sentence ‘The units are specified by the dfdl:lengthUnits property’ and replace with the sentence ‘For dfdl:lengthKind 'delimited', 'pattern' and 'endOfParent' the length units are always characters, for other dfdl:lengthKinds the length units are specified by the dfdl:lengthUnits property.’

2.131. *Section 12.3.3.* After Table 15 add that it is a schema definition error if type is xs:string and lengthKind is 'implicit' and lengthUnits is 'bytes' and encoding is not an SBCS encoding. This prevents a scenario where validation against maxLength facet is in characters but parsing and unparsing using maxLength facet is in bytes.

2.132. *Section 12.3.7.* In the paragraph that discusses specified length elements that are considered to have variable length when unparsing, add that it is a schema definition error for such elements if type is xs:string and textPadKind is not 'none' and lengthUnits is 'bytes' and encoding is not an SBCS encoding and minLength facet is not zero. This prevents a scenario where validation against minLength facet is in characters but padding to minLength facet is in bytes.

2.133. *Section 13.11.1.* For the calendar pattern symbol ‘I’ add that the omission of time zone from the input data when the type is xs:dateTime or xs:time is not a processing error. If that occurs then the time zone is obtained from the calendarTimeZone property.

2.134. *Section 3.* For the specification to correctly discuss parsing and unparsing of character data, the following new terms are added to the Glossary, and used in appropriate places in the rest of the spec.

- CCSID - see *Coded Character Set Identifier*

- Character - A ISO10646 character having a unique *character code* as its identifier. This concept is independent of font, typeface, size, and style, so '**F**', '**F**', '*F*', are all the same character 'F'

- Character Code - The canonical integer used to identify a character in the ISO10646 standards. This number identifies the character, but can be independent of any specific character set encoding of the character. Example: The '{' character known in Unicode as LEFT CURLY BRACKET. Has character code U+007B. However, depending on the *character set encoding*, the value 0x7B may or may not appear in the representation of that character.

- Character Set - An abstract set of characters that are assigned (or *mapped to)* a representation by a particular *character set encoding*. For most character set encodings their character set is a subset of the Unicode character set.

- Character Set Encoding - Often abbreviated to just 'encoding'. A specific representation of a character set as bytes or bits of data. A character set encoding is usually identified by a standard character set encoding name or a recognized alias name, or by a *coded character set identifier or CCSID*. These identifiers are standardized. The names and aliases are standardized by the IANA (where unfortunately, they are called character set names). CCSIDs are an industry standard. Examples of character set encoding names are UTF-8, USASCII, GB2312, ebcdic-cp-it, ISO-8859-5, UTF-16BE, Shift_JIS. The DFDL standard allows for implementation-specific character set encodings to be supported, and standardizes one name that is DFDL-specific which is USASCII-7bit-packed.

- Character Width - The number of code units or alternatively the number of bytes used to represent a character in a specific character set encoding is called the character width. Encodings are either fixed width (all characters encoded using the same width), or variable-width (different characters are encoded using different widths). For example the UTF-32 character set encoding has 4-byte character width, whereas USASCII has a 1-byte character width. UTF-8 is variable width, and any specific character has width 1, 2, 3, or 4 bytes.

- Code Unit - When a character set encoding uses differing *variable width* representations for characters, the units making up these variable width representations are called *code units*. For example the UTF-8 encoding uses between 1 and 4 code units to represent characters, and for UTF-8, the individual code units are single bytes. DFDL's interpretation of the UTF-16 encoding is either fixed or variable width. When format property dfdl:utf16Width='variable' then UTF-16 is variable width and this encoding uses either one or two code units per character, but in this case each individual code unit is a 16-bit value. When a character set is fixed width, then there is no distinction between a code unit and a code point.

- Coded Character Set Identifier (CCSID) - An alternate identifier of a character set encoding. Originally created by IBM, CCSIDs are a broadly used industry standard.

- Encoding - See *Character Set Encoding*

- Fixed-Width Character Encoding - A character set encoding where all characters are encoded using a single code unit for their representation. Note that a code unit is not necessarily a single byte.

- Surrogate Pair - A Unicode character whose character code value is greater than 0xFFFF can be encoded into variable-width UTF-16BE or UTF-16LE (which are variable-width encodings when the DFDL property utf16Width='variable'). In this case the representation uses two adjacent *code units* each of which is called a surrogate, and the pair of which is called a surrogate pair.

- Unicode - A character set defined by the Unicode Consortium, and standardized at the International Standards Organization (ISO) as ISO10646.

- Variable-Width Character Encoding - A character set encoding where characters are encoded using one or more code units for their representation depending on which specific character is being encoded. An example is UTF-8 which uses from 1 to 4 bytes to encode a character.

2.135. *Section 23.5.* State the types of arguments and return values where not specified.

- *23.5.2.1.* The return value of each Boolean function is xs:boolean.

- *23.5.2.4.* The return value of each Date, Time function is xs:integer except fn:seconds-from-dateTime and fn:seconds-from-time which return xs:decimal.
- *23.5.6.* The return value of fn:local-name is changed to xs:string.
- *23.5.3.* The $lengthUnits argument of dfdl:contentLength and dfdl:valueLength is xs:string.
- *23.5.3.* The $data argument of dfdl:testBits is xs:unsignedByte.
- *23.5.3.* The $bitPos argument of dfdl:testBits is xs:nonNegativeInteger.

2.136. *Section 23.5.3.* Three new DFDL specific functions are provided that return the timezone from a calendar type. These complement the XPath functions that return other calendar components from calendar types.

dfdl:timeZoneFromDateTime ($arg) dfdl:timeZoneFromDate ($arg) dfdl:timeZoneFromTime ($arg)	Returns the timezone component of $arg if any. If $arg has a timezone component, then the result is a string in the format of an ISO Time zone designator. Interpreted as an offset from UTC, its value may range from +14:00 to -14:00 hours, both inclusive. The UTC time zone is represented as "+00:00". If the $arg has no timezone component, then "" (empty string) is returned.

2.137. *Section 13.11.1.* Correct the paragraph for fractional seconds to say that excess fractional seconds are truncated, and not rounded up. (This is to match ICU behaviour.)

2.138. *Section 12.3.7.* When representation is binary and the length specified for an element implies that the capacity of the simple type may be exceeded, the behaviour of the DFDL processor is not consistent and is dependent on whether lengthUnits is ‘bits’ or ‘bytes’. This is addressed. It is still a schema definition error if the length of a bit field is too large for the corresponding integer type when statically verifiable, but it should be a processing error if it occurs at runtime, and not a runtime schema definition error as stated. The same rules should also be applied when lengthUnits is 'bytes'.

2.139. *Section 12.3.7.2.* Clarify that numbers with a binary packed representation are allowed to have lengthUnits ‘bits’ but the length must be a multiple of 4 and it is a schema definition error otherwise.

2.140. *Section 12.1.* Clarify that numbers with a binary packed representation must be aligned on a nibble (ie, 4-bit) boundary and it is a schema definition error otherwise.

2.141. *Section 13.11.* Change the type of property calendarLanguage so that it is String or DFDL Expression. If an expression is provided, it must return a string that complies with the pattern given by errata 2.44. This enhancement allows DFDL schemas to be authored that model locale-dependent calendars.

2.142. *Section 11.* Clarify that property ignoreCase plays no part when comparing an element value with an XSDL enum facet, matching an element value to an XSDL pattern facet, or comparing an element value with the XSDL fixed property. It is therefore not used by validation when enabled, nor by the dfdl:checkConstraints function.

2.143. *Section 12.3.5.1.* For lengthKind ‘pattern’ clarify that when a DFDL regular expression is matched against data:

- The data is decoded from the specified encoding into Unicode before the actual matching takes place.
- If there is no match (ie, a zero-length match) it is not a processing error but instead it means the length is zero.

2.144. *Section 7.3.1 and 7.4.1.* Allow the message property of an assert or a discriminator to be either a string or a DFDL Expression that returns a string.

Any element referred to by the message expression must have already been processed or must be a descendent of the component carrying the assert or discriminator (same rule as for the test expression).

Example:
```
<dfdl:assert message="{ fn:concat('unknown whatever ', ../data1) }">
{  if (...pred1...) then ...expr1...
   else if (...pred2...) then ...expr2...
   else fn:false()
 }</dfdl:assert>
```

The message specified by the message property is issued only if the assert or discriminator is unsuccessful, that is, the test expression evaluates to false or the test pattern returns a zero-length match. If so, and the message property is an expression, the message expression is evaluated at that time.

If a processing error or schema definition error occurs while evaluating the message expression, a recoverable error should be issued to record this error, then processing of the assert or discriminator continues as if there was no problem and in a manner consistent with the failureType property, but using an implementation-defined substitute message.

2.145. *Section 6.3.* The specification does not formally state the XSDL type of all the DFDL property types. That is corrected as follows:

- DFDL string literal: restriction of xs:token that disallows the space character.
- DFDL expression : xs:string
- DFDL regular expression : xs:string
- Enumeration: xs:token

In addition:

- Leading/trailing spaces are trimmed for DFDL expressions
- Leading/trailing spaces are not trimmed for DFDL regular expressions

2.146. *Section 23.5.3.* XPath 2.0 is not very good with literal hex binary data, in that the only types you can create are xs:hexBinary and xs:string. There is sometimes a need to create a number type from hex binary, and a hex binary type from a number. Accordingly the following new DFDL specific functions are added.

dfdl:byte ($arg) dfdl:unsignedByte ($arg) dfdl:short ($arg) dfdl:unsignedShort ($arg) dfdl:int ($arg) dfdl:unsignedInt ($arg) dfdl:long ($arg) dfdl:unsignedLong ($arg)	These constructor functions behave identically to the XPath 2.0 constructor functions of the same names, with one exception. The argument can be a quoted string beginning with the letter 'x', in which case the remainder of the string is hexadecimal digits that represent a big-endian twos complement representation of a binary number. If the string begins with 'x', it is a schema definition error if a character appears other 0-9, a-f, A-F.

	Each constructor function has a limit on the number of hex digits, with no more digits than 2, 4, 8, or 16 for the byte, short, int and long versions respectively. It is a schema definition error if more digits are encountered than are suitable for the type being created

Examples:

- dfdl:unsignedInt("xa1b2c3d4") is the unsigned int value 2712847316.
- dfdl:int("xFFFFFFFF") is the signed int value -1.
- dfdl:unsignedByte("xFF") is the unsigned byte value 255.
- dfdl:byte("xff") is the signed byte value -1.
- dfdl:byte("x7F") is the signed byte value 127.
- dfdl:byte("x80") is the signed byte value -128.
- dfdl:unsignedByte("x80") is the unsigned byte value 128.
- dfdl:byte("x0A3") is a schema definition error (too any digits for type).
- dfdl:byte("xG3") is a schema definition error (invalid digit).

dfdl:hexBinary ($arg)	This constructor function behaves identically to the XPath 2.0 constructor function of the same name, with one exception. The argument can also be a long, unsignedLong, or any subtype thereof, and in that case a xs:hexBinary value containing a number of hex digits is produced. The ordering and number of the digits correspond to a binary big-endian twos-complement implementation of the type of the argument. Digits 0-9, A-F are used. The number of digits produced depends on the type of $arg, being 2, 4, 8 or 16. If $arg is a literal number then the type is the smallest signed type (long, int, short, byte) that can contain the value. If a literal number is not able to be represented by a long, it is a schema definition error.

Examples:

- dfdl:hexBinary(xs:short(208)) is the hexBinary value "00D0".
- dfdl:hexBinary(208) is the hexBinary value "D0".
- dfdl:hexBinary(-2084) is the hexBinary value "F7FF".

2.147. *Section 23.1*. Replace the paragraphs that talk about allowable element references in DFDL expression paths when parsing and unparsing with the following paragraph.

In general, a DFDL expression can reference any element that precedes the position in the schema where the expression is declared, with the following exceptions:

- An assert or discriminator on a component may reference an element that is a descendent of the component.
- A dfdl:outputValueCalc property may reference an element that follows the position in the schema where the property is specified.
- It is a schema definition error if a component in a choice branch references an element in another branch of the same choice or a descendent of such an element

- It is a schema definition error if an element in an unordered sequence group references an element in the same sequence group or a descendent of such an element.
- It is a schema definition error if an element in an ordered sequence group references a floating element in the same sequence group or a descendent of such an element.

2.148. *Section 12.2*. Clarify DFDL Character Class entities allowed in delimiters.

The initiator, terminator, and separator properties can have the character class entities NL, WSP, WSP+, WSP*, but not WSP* on its own. They cannot have ES.

2.149. *Section 21*. The raw byte entities feature is added to the list of optional features in the standard.

2.150. *Section 13.11.1.* Property calendarPattern: Add support for calendar pattern 'EEEEEE' (6 x 'E') and 'eeeeee' (6 x 'e') provided by ICU, which provide a 2 letter abbreviation, eg, 'Mo'.

The 'EEEEE' (5 x 'E') form is broken in some versions of the ICU library. Implementations should either fix this or release-note the limitation. The DFDL specification includes the 'EEEEE' functionality as specified by ICU, irrespective of any bugs/flaws in ICU library versions.

In general, flaws in the ICU libraries, or inconsistencies between the ICU4C and ICU4J variants of this library are not issues that affect the DFDL specification, but rather are limitations of implementations and should be release-noted or otherwise called out by implementations so that users can understand their impact.

2.151. *Section 13.11*. Property calendarCheckPolicy: Clarify strict and lax behaviour as follows:

<u>1) Lenient parsing behaviour when in 'strict' mode:</u>
a) case insensitive matching for text fields
b) MMM, MMMM, MMMMM all accept either short or long form of Month
c) E, EE, EEE, EEEE, EEEEE, EEEEEE all accept either abbreviated, full, narrow and short forms of Day of Week
d) accept truncated leftmost numeric field (eg, pattern "HHmmss" allows "123456" (12:34:56) and "23456" (2:34:56) but not "3456")

<u>2) Additional lenient parsing behaviour when in 'lax' mode:</u>
a) values outside valid ranges are normalized (eg, "March 32 1996" is treated as "April 1 1996")
b) ignoring a trailing dot after a non-numeric field
c) leading and trailing whitespace in the data but not in the pattern is accepted ****
d) whitespace in the pattern can be missing in the data
e) partial matching on literal strings (eg, data "20130621d" allowed for pattern "yyyyMMdd'date' " ****

**** Only in ICU4C as of ICU 51. ICU4J will be changed to match ICU4C. Implementations are advised to document this limitation with a release note if it affects their functionality.

2.152. *Section 14.4.* Clarifications around sequences containing floating elements.

A non-floating array element must have its occurrences appearing contiguously, so the floating element can't appear in-between. In other words, floating 'yes' only makes a statement about the floating element, not about any other elements in the sequence.

Change wording to:

"An ordered sequence of n element children all with dfdl:floating='yes' is equivalent to an unordered sequence with the same n element children with dfdl:floating='no'."

Add restrictions:

It is a schema definition error if an element with dfdl:floating 'yes' is an optional element or an array element and its dfdl:occursCountKind property is not 'parsed'.

It is a schema definition error if two or more elements with dfdl:floating 'yes' in the same group have the same name and the same namespace.

2.153. *Section 3, 12.3.7.2*. Clarify length of elements with binary representation. Separate the material about computing the values of elements of binary representation.

- Move to glossary entries, moving them out of the sections on length.
    - Bit Position
    - Bit String
- Add new glossary entries for these terms, which we use repeatedly.
    - Data Stream
    - Binary - clarify ambiguity around binary meaning not text, and binary meaning twos-complement.
    - Decimal - clarify ambiguity around decimal meaning base-10, and decimal meaning binary packed representations.
    - Text
    - Twos-Complement

- Change titles of section 12.3.7.1 "… with dfdl:representation 'text', to "…with textual representation"
- New section title for 12.3.7.2, like 12.3.7.1, but "… with binary representation".
- Move materials on computing values of binary integers to section 13.7.1.

2.154. *Section 13.11*. Property calendarLanguage. Add statement about required language support.

All DFDL Implementations must support calendarLanguage value "en". Implementations may support additional values, however, the values are always interpreted as a Unicode Language Identifier as defined by the Unicode Locale Data Markup Language [ULDML] and the Unicode Common Locale Data Repository [UCLDR]. These references are added to the references section of the spec.

2.155. *Sections 3, 7.3.1, 7.3.2, 12.3.5*. Scan, scannable, scannable-as-text
These terms all added to the glossary. Definitions removed from the prose. Scannable now means able to scan, which is natural. More specific term scannable-as-text used when we want the recursive requirement of uniform encoding.

Errata 2.9 updated to use term scannable-as-text.

2.156. *Section 13.6.1*. Remove the following statement:

"If the pattern uses digits/fractions then these must match any XML schema facets. If not it is a schema definition error."

2.157. *Section 9.2*. Definition of grammar construct RightPadOrFill is not correct.

There is a possibility that both padding and filling can occur on the right of a text element with specified length in bytes, a non-SBCS encoding and textPadKind 'padCharacter'. This occurs when the specified length does not exactly match the encoded length including padding. This gap is filled with the fillByte.

The updated grammar is in Chapter 4 of this document.

3. Major Errata

The following major errata have been identified.

3.1. *Section 14.5.* Changes to placement of property hiddenGroupRef.

Change to behave like the ref property. That is, it cannot be placed in scope by a format annotation, and is only set at its point of use. Empty string is not an allowed value. This reflects that there is no hiddenGroupRef value that applies universally.

The spec is not clear as to whether this property is allowed on the sequences that are the direct children of global groups, or on group references. Clarify that it is allowed on any xs:sequence but not on any xs:group, including group reference, nor can it appear on any xs:choice.

If hiddenGroupRef appears on a sequence, the appearance of any other DFDL properties on that sequence is a schema definition error.

3.2. *Section 17.* Changes to placement of properties inputValueCalc and outputValueCalc.

Change to behave like the ref property. That is, they cannot be placed in scope by a format annotation, and are only set at their point of use. Empty string is not an allowed value. This reflects that there is no inputValueCalc or outputValueCalc property value that applies universally

The spec is confused as to whether these properties are applicable to simple types. Remove any references to these properties in relation to simple types, as they are applicable to elements only. Any application to simple types is a future extension.

The spec is not clear as to whether these properties are allowed on global elements or element references. Clarify that they are allowed on local element and element references but not on global elements.

Add that inputValueCalc is not allowed to appear on a local element or element reference that is the root of a choice branch.

If inputValueCalc appears on an element, the appearance of any other DFDL properties on that element is a schema definition error.

3.3. *Section 12.3.* Clarify that when property is lengthKind 'explicit', 'implicit' (simple only), 'prefixed' or 'pattern', it means that delimiter scanning is turned off and in-scope delimiters are not looked for within or between elements.

Consequently remove the last paragraph of section 5.2.2 starting "It is a processing error when a fixed-length string is found to have a number of characters not equal to the fixed number".

3.4. *Sections 2 and 7.3.* Add a new failure type ‘recoverable error’ for use by the assert annotation when parsing, to permit the checking of physical constraints without terminating a parse. For example, using an assert to check a physical length constraint when property lengthKind is 'delimited'. Details:

- After a recoverable error the parser will continue.
- Importantly, it does not cause backtracking to take place when speculating.

- It can be raised via a new enum attribute on dfdl:assert called 'failureType'.
- An error occurring during evaluation of a dfdl:assert remains a processing error.
- All existing stated processing errors remain as such.
- Discriminators remain unchanged.
- The issuing of recoverable errors is independent of whether validation is enabled.

Property Name	Description
failureType	Enum (optional) Valid values are 'processingError', 'recoverableError'. Default value is 'processingError'. Specifies the type of failure that occurs when the dfdl:assert is unsuccessful. When 'processingError', a processing error is raised. When 'recoverableError', a recoverable error is raised. Annotation: dfdl:assert

Considered extending validation error to cover this, but the spec is quite clear that a validation error is a logical check performed on the infoset and the behaviour of the DFDL processor is unspecified.

3.5. *Section 13.8.* The spec is not clear which variants of IEEE binary floats are supported. Clarify that support is for IEEE 754-1985, the same as XSDL 1.0. The implications of this are:

- xs:float must have a physical length of 4 bytes for both 'ieee' and 'ibm390Hex' (schema definition error if explicit length is other than 4).
- xs:double must have a physical length of 8 bytes for both 'ieee' and 'ibm390Hex' (schema definition error if explicit length is other than 8).
- Add statement that there may be precision/rounding issues when converting IBM float/double to/from infoset float/double which is IEEE
- Half-precision IEEE and quad-precision IEEE/IBM are not supported

Noted that XSDL 1.1 moved to IEEE 754-2008 only because of new decimal support, and not for enhanced float support. That's why in XSDL 1.1 there are still just the xs:float and xs:double built-in types. Any future support for half-precision and quad-precision in XSDL would very likely be implemented by adding new built-in types that derive from xs:anySimpleType. It is likely therefore that future DFDL support for half-precision and quad-precision will build on XSDL.

3.6. *Section 4.* It was observed that the content of the DFDL infoset after parsing is not sufficient to build a W3C Post Schema Validation infoset (PSVI). Specifically, two things are missing:

- whether an element is valid
- for a simple element with a union type, which member the value matched.

In order to achieve this the DFDL infoset is modified as follows:

- Add a new Boolean **[valid]** member to element information item. A complex element information is not valid if any of its [children] are not valid. Empty if validation is not enabled.
- Add a new string **[unionMemberSchema]** member to simple element information item. This is an SCD reference to the member of the union that matched the value of the element. Empty if validation is not enabled. Empty if the element’s type is not a union.

On unparsing, any non-empty values for these properties are ignored. However, the augmented infoset which is built from the unparse operation should contain values for these properties if validation is enabled during unparsing.

3.7. *Section 4, 9, 11, 12.3.7.1.3.* Forcing a DFDL author to explicitly model a Unicode byte order mark (BOM) is a significant usability issue. Most authors working with Unicode data will expect a DFDL processor to handle BOMs in the same way as other software applications. Accordingly the DFDL specification is enhanced to add automatic detection and generation of Unicode BOMs.

A new string **[unicodeByteOrderMark]** member is added to the DFDL infoset document information item. When the encoding of the root element of the document is exactly UTF-8, UTF-16, or UTF-32 (or CCSID equivalent), the member value indicates whether the document starts with a BOM. If there is a BOM then for UTF-8 encoding the value is 'UTF-8'; for UTF-16 encoding the value is 'UTF-16LE' or 'UTF-16BE'; for UTF-32 the value is 'UTF-32LE' or 'UTF-32BE'. If there is no BOM then the member value is empty. When the encoding of the root element of the document is any other encoding, the member value is empty.

The grammar production for the overall document changes to accommodate a BOM as shown in Chapter 4 of this document.

Parsing behaviour: When the dfdl:encoding property of the root element is specified, and is exactly one of UTF-8, UTF-16, or UTF-32 (or CCSID equivalents), then a DFDL parser will look for the appropriate BOM as the very first bytes in the data stream.

- UTF-8. If a BOM is found then this is used to set the document information item [unicodeByteOrderMark] member. If no BOM is found the parser takes no action. There is no need to model the BOM explicitly.

- UTF-16. If a BOM is found then this is used to set the document information item [unicodeByteOrderMark] member, and all data with dfdl:encoding UTF-16 throughout the rest of the stream are assumed to have the implied byte order. If no BOM is found then all data with dfdl:encoding UTF-16 throughout the rest of the stream are assumed to have big-endian byte order. There is no need to model the BOM explicitly.

- UTF-32. If a BOM is found then this is used to set the document information item [unicodeByteOrderMark] member, and all data with dfdl:encoding UTF-32 throughout the rest of the stream are assumed to have the implied byte order . If no BOM is found then all data with dfdl:encoding UTF-32 throughout the rest of the stream are assumed to have big-endian byte order. There is no need to model the BOM explicitly.

When the dfdl:encoding property of the root element is specified, and is exactly one of UTF-16LE, UTF-16BE, UTF-32LE or UTF-32BE (or CCSID equivalents), then a DFDL parser will **not** look for the appropriate BOM. The byte order to use is implicit in the encoding. If a BOM does appear at the start of the data stream, then it simply will be treated as a Unicode Zero-Width Non-Breaking Space (ZWNBS) character, because this shares the same codepoint as a BOM.

The dfdl:byteOrder property is never used to establish the byte order for Unicode encodings.

The parser never looks for a BOM at any other point in the data stream, so if a BOM appears elsewhere it will be treated as a Unicode ZWNBS character as described above.

Unparsing behaviour: When the dfdl:encoding property of the root element is specified, and is exactly one of UTF-8, UTF-16 or UTF-32 (or CCSID equivalents), then a DFDL unparser will look in the infoset document information item for a BOM.

- UTF-8. If the document information item [unicodeByteOrderMark] member is 'UTF-8', the UTF-8 BOM is output as the very first bytes in the data stream. If the property is empty then no BOM is output. If the property has any other value, it is a processing error. There is no need to model the BOM explicitly.

- UTF-16. If the document information item [unicodeByteOrderMark] member is 'UTF-16LE' or 'UTF-16BE', the corresponding UTF-16 BOM is output as the very first bytes in the data stream, and all data with dfdl:encoding UTF-16 throughout the rest of the document will be output with the implied byte order. If the property is empty then no BOM is output, and all data with dfdl:encoding UTF-16 throughout the rest of the document are assumed to have big-endian byte order. If the property has any other value, it is a processing error. There is no need to model the BOM explicitly.

- UTF-32. If the document information item [unicodeByteOrderMark] member is 'UTF-32LE' or 'UTF-32BE', the corresponding UTF-32 BOM is output as the very first bytes in the data stream, and all data with dfdl:encoding UTF-32 throughout the rest of the document will be output with the implied byte order . If the property is empty then no BOM is output, and all data with dfdl:encoding UTF-32 throughout the rest of the document are assumed to have big-endian byte order. If the property has any other value, it is a processing error. There is no need to model the BOM explicitly.

When the dfdl:encoding property of the root element is specified, and is exactly one of UTF-16LE, UTF-16BE, UTF-32LE or UTF-32BE (or CCSID equivalents), then a DFDL unparser will **not** look at the document information item [unicodeByteOrderMark] member and will **not** output a BOM. The byte order to use is implicit in the encoding. If a BOM does need to be output at the start of the data stream, then it must be explicitly modelled as such.

The dfdl:byteOrder property is never used to establish the byte order for Unicode encodings. The unparser never outputs a BOM at any other point in the data stream. If a BOM needs to appear, then it must be explicitly modelled as such.

3.8. *Section 2.2*. Clarification is needed to schema definition error reporting criteria. The intent of the spec is that a DFDL processor only needs to report schema definition errors that directly affect its processing of the data. This is because the nature of DFDL's scoping rules mean that often it is not possible to validate an object definition for correctness in isolation.

Clarify that a DFDL processor:

- That only implements a DFDL parser does not have to validate properties that are solely used when unparsing, though it is recommended that it does so for portability reasons.
- That does not implement some optional features does not have to validate properties or annotations required by those optional features, but MUST issue a warning that an unrecognized property or annotation has been encountered.
- Need not validate global objects as they may legitimately be incomplete, with the following exceptions which must be validated:
    1. Global simple types that are referenced by prefixLengthType property
    2. Global elements that are the document root.

Clarify what action a DFDL processor should take when it encounters an object that explicitly carries properties that are not relevant to the object as defined.

- Property not applicable to the object's DFDL annotation.
  Schema definition error. Example is lengthKind on xs:sequence.
- Property not applicable because of simple type.
  Warning (optional). Example is calendarPatternKind on xs:string.
- Property not applicable because of another DFDL property setting.
  Warning (optional). Example is binaryNumberRep when representation is text.

3.9. *Section 12.3.5*, *7.3.1, 7.3.2.* The spec originally allows lengthKind 'pattern' to be used when the representation of the current element, or of a child element, is binary, but imposes restrictions on the encoding that can be in force.

Clarify that the encoding property must be defined for the element (else schema definition error), and that a decoding processing error is possible if the match of the regex encounters data that does not decode in that encoding, dependent on the setting of encodingErrorPolicy. Remove section 12.3.5.1.

Same clarifications needed for testKind "pattern" property for asserts and discriminators.

For consistency, the restriction that a complex element of specified length and lengthUnits 'characters' must have children that are all text and that have the same encoding as the complex element, is dropped.

3.10. *Sections 5.1, 13.15.* Allow complex elements to be nillable. There are advantages in permitting complex elements to be nillable as well as simple elements. For example, it provides better interoperability with XML infosets. However, to avoid the concept of a complex element having a value, which is not possible in DFDL, the only permissible nil value is the empty string, represented by the DFDL %ES; entity.

If a complex element has xs:nillable set to 'true', it is a schema definition error if nilKind is not 'literalValue' or nilValue is not the single value '%ES;'.

Allowing complex elements to be nillable also solves another problem, that of preserving the position of optional complex elements in an array that contains explicit gaps. An infoset item with the special value nil is created for each such gap.

Property nilValueDelimiterPolicy is applicable.

The grammar changes to reflect this, as shown in Chapter 4 of this document.

3.11. *Section 16.* If the occurrences of an element in the data are not fixed (that is, the element is a variable array or is optional) and the count of the number of elements is not provided in the data nor is there a stop value, then the DFDL language only provides one mechanism for deducing the number of elements when parsing, namely occursCountKind 'parsed'. This causes the parser to speculate indefinitely until no more elements can be established. However there are circumstances where the minimum and maximum number of elements is known, and these facts could be used to guide the parse.

A new occursCountKind enumeration called 'implicit' is added, which takes into account minOccurs and maxOccurs settings.

The descriptions of the behaviour for the all occursCountKind enums is greatly enhanced, to cover both parsing and unparsing, to provide a rewrite semantic for an array as a sequence, to introduce a forward progress requirement and to clarify the action taken for non-normal representations.

This is covered in a separate DFDL experience document [DFDLX2].

3.12. *Section 2.4*. Validation checks are constraints expressed in XSDL, and they apply to the logical content of the infoset. Originally the spec says 'an unparse validation error occurs when the physical representation being output would generate a validation error when parsing the data representation using the same DFDL schema.' This is a convenient definition, but problematic, because the original infoset used by the unparser could have been invalid, and the act of DFDL unparsing created a data stream which when parsed created a valid infoset. This can occur because of rounding, for example.

The specification will be changed to say that validation on parsing takes place on the infoset that is created by the parse, and that validation on unparsing takes place on the *augmented* infoset that is created by the unparser as a side-effect of creating the output data stream.

The new approach is in keeping with the way that XML Schema 1.0 defines validation against its PSVI.

3.13. *Sections 4.1.2, 11.* DFDL does not adequately describe how to handle decoding and encoding errors.

A new sub-section is added to section 11. *(this is probably 11.2, if 11.1 is about Unicode byte order marks)*

11.2 Character Encoding and Decoding Errors
When parsing, these are the errors that can occur when decoding characters into Unicode/ISO 10646.

1. The data is broken - invalid bit/byte sequences are found which do not match the definition of a character for the encoding.
2. Not enough data is found to make up the entire encoding of a character. That is, a fragment of a valid encoding is found.

When unparsing, these are the errors that can occur when encoding characters from Unicode/ISO 10646 into the specified encoding.
1. No mapping provided by the encoding specification.
2. Not enough room to output the entire encoding of the character (e.g., need 3 bytes for a character encoding that uses 3-bytes for that character, but only 1 byte remains in the available length.
The subsections below describe how these errors are handled.

11.2.1 property dfdl:encodingErrorPolicy

A new property dfdl:encodingErrorPolicy is added.

Property Name	Description
encodingErrorPolicy	Enum Valid values are 'error', 'replace'. Specifies the action to take when a character decoding error occurs when parsing or a character encoding error occurs when unparsing. Applies whenever dfdl:encoding is used. When 'error', a processing error is raised. When 'replace', a substitution character is used if one is available. See section 11.2 for full description. Annotation: dfdl:element, dfdl:simpleType, dfdl:sequence, dfdl:choice, dfdl:group

11.2.1.1 dfdl:encodingErrorPolicy='error'

If 'error', then any error when decoding characters while parsing causes a parse error. For unparsing, any error when encoding characters causes an unparse error.

When parsing, it does not matter if this happens when scanning for delimiters, matching a regular expression, matching a literal nil value, or constructing the value of a textual element.

There is one exception. When lengthUnits='bytes', the 'not enough data' decode error is ignored, and the data making up the fragment character is skipped over. Symmetrically, when unparsing the 'not enough room' encoding error is ignored and the left-over bytes are filled with the dfdl:fillByte.

11.2.1.2 dfdl:encodingErrorPolicy='replace' for Parsing

If 'replace' then any error results in the insertion of the Unicode Replacement Character (U+FFFD) as the replacement for that error.
It does not matter if this error and replacement happens when scanning for delimiters, matching a regular expression, matching a literal nil value, or constructing the value of a textual element.

There is one exception. When lengthUnits='bytes', the 'not enough data' decode error is ignored, no replacement character is created. The data making up the fragment character is skipped over. (It will be filled with the dfdl:fillByte when unparsing.)

Note that the "." wildcard in regular expressions will match the Unicode Replacement Character, so ".*" and ".+" regular expressions can potentially cause very large matches (up to the entire data stream) to occur when data contains errors and dfdl:encodingErrorPolicy='replace'. Bounded length negated regular expressions can help in this case. E.g., " [^\uFFFD]{0,50}" says to match any character excluding Unicode Replacement Characters, but only up to length 50.

It is also worth noting that the Unicode Replacement Character can appear in data as an ordinary character, and this cannot be distinguished from the insertion of the Unicode Replacement Character due to a decode error.

If lengthUnits='characters', then a Unicode Replacement Character counts as contributing a single character to the length.

If the data contains more than one adjacent decode error, then the specific number of Unicode Replacement Characters that are inserted as the replacement of these errors is implementation dependent. That is, some implementations may view , for example, three consecutive erroneous bytes as three separate decode errors, others may view them as a single or two decode errors. All implementations MUST, however, insert some number of Unicode Replacement Characters, and then continue to decode characters following the erroneous data.

The trimming of padding characters always happens after Unicode Replacement Characters have been inserted into the data.

11.2.1.3 dfdl:encodingErrorPolicy='replace' for Unparsing

For unparsing, each encoding has a replacement/substitution character specified by the ICU. This character is substituted for the unmapped character or the character that has too large an encoding to fit in the available space.

There is one exception. When lengthUnits='bytes', the 'not enough room' encoding error is ignored. The left-over bytes are filled with the dfdl:fillByte (they are skipped when parsing.)

The definitions of these substitution characters can be conveniently found for many encodings in the ICU Converter Explorer (http://demo.icu-project.org/icu-bin/convexp).

An encoding error is an unparse error if the encoding does not provide a substitution/replacement character definition. (This would be rare, but could occur if a DFDL implementation allows many encodings beyond the minimum set.)

11.2.1.4 Parsing: Unicode Decoding Non-Errors

The following specific situations involving encodings UTF-16, UTF-16LE, and UTF-16BE when utf16Width="fixed", and they do not cause a decoding or encoding error.

- unpaired surrogate code-point
- out-of-order surrogate code-point pair
- surrogate code point pair is encountered

In all these cases the code-point(s) becomes a character code in the DFDL Information Item for the string.

11.2.2 Preserving Data Containing Decoding Errors

There can be situations where data wants to be preserved exactly even if it contains errors. It is suggested that if a DFDL schema author wants to preserve information containing data where the data may have decoding errors, that they model such data as xs:hexBinary, or as xs:string but using an encoding such as iso-8859-1 which preserves all bytes.

3.14. *Section 14.2.* To better describe the property and its behaviour, property separatorPolicy is renamed to separatorSuppressionPolicy, and its enums renamed as follows:

'required' -> 'never'
'suppressed' -> 'anyEmpty'
'suppressedAtEndLax' -> 'trailingEmpty'
'suppressedAtEndStrict -> 'trailingEmptyStrict'.

Additionally the property description for separatorSuppressionPolicy is rewritten, introductory paragraphs are added to section 14.2, and section 14.2.1 is replaced with new tables.

This is covered in a separate DFDL experience document [DFDLX2].

3.15. *Section 15.* A new mechanism is introduced for resolving choices, the motivation being to make the cost of resolution close to constant time for choices with large numbers of branches where the branch to take is known in advance of parsing the choice.

A new element property is added called choiceBranchKey of type 'DFDL String Literal'. This provides an alternative way to discriminate a choice containing this element. Allowed on local element and element reference only.

A new dfdl:choice property is added called choiceDispatchKey of type 'DFDL Expression'. The expression must evaluate to an xs:string. The resultant string must match (case insensitive) the choiceBranchKey property value of one of the element branches of the choice, and if so discriminates in favour of that branch. The parser then goes straight to that branch, ignoring schema order.

Rules:

Because the branch is 'known to exist' no backtracking takes place if a processing error subsequently occurs.

Both properties are non-representation properties (see erratum 2.126), it is not possible to set a value in scope by a dfdl:format annotation, and a value can only set at its point of use. This is because there is nothing sensible that could be set in scope. Empty string is not an allowed value.

Both properties are only used when parsing.

When choiceDispatchKey is present, all choice branches must be local elements or element references. It is a schema definition error otherwise.

It is a processing error if the resolved value of choiceDispatchKey does not match one of the choiceBranchKey values.

It is a schema definition error if individual choiceBranchKey values are not unique across all elements that are branches of a choice that carries choiceDispatchKey

It is a schema definition error if both initiatedContent and choiceDispatchKey are provided on the same choice.

It is not a schema definition error if either initiatedContent or choiceDispatchKey is provided on a choice and a discriminator exists on a choice branch. In this case the discriminator will apply to a point of uncertainty that encloses the choice.

DFDL entity character classes and DFDL raw byte entities are not allowed in choiceBranchKey.

3.16. *Section 14.2*. Property documentFinalSeparatorCanBeMissing is removed as it is redundant. A postfix separator where the final separator can be missing can be modelled as an infix separator with documentFinalTerminatorCanBeMissing on the parent element.

3.17. *Section 21*. The list of optional DFDL features is extended to make it easier for implementers to create minimal and extended conforming DFDL processors.

Feature	**Detection**
Text representation for types other than String	dfdl:representation="text" for Number, Calendar or Boolean types
Delimiters	dfdl:separator <> "" or dfdl:initiator <> "" or dfdl:terminator <> "" or dfdl:lengthKind="delimited"
BCD calendars	dfdl:binaryCalendarRep="bcd"
BCD numbers	dfdl:binaryNumberRep="bcd"
Multiple schemas	xs:include or xs:import in xsd
Named Formats	dfdl:defineFormat or dfdl:ref
Choices	xs:choice in xsd
Arrays where size not known in advance	dfdl:occursCountKind 'implicit', 'parsed', 'stopValue'
Expressions	Use of a DFDL expression in any property or attribute value
End of parent	dfdl:lengthKind = "endOfParent"
IBM 4690 packed calendars	dfdl:binaryCalendarRep="ibm4690Packed"
IBM 4690 packed numbers	dfdl:binaryNumberRep="ibm4690Packed"
DFDL Byte Value Entities	Use of %#r syntax in a DFDL String Literal

Existing optional feature 'Variables' is clarified to be dependent on optional feature 'Expressions'.

3.18. *Section 9.2, 23.5.3.* The DFDL grammar productions are revised to make clear the distinction between the different allowable representations that an element can have and to enforce the correct use of the terms 'content', 'value' and 'representation'.

This has a significant effect on the grammar is shown in Chapter 4 of this document.

All sections of the specification are updated to ensure that 'content', 'value' and 'representation' are used correctly and consistently.

As a consequence two of the DFDL-specific functions are renamed:
dfdl:representationLength() -> dfdl:contentLength()
dfdl:unpaddedLength() -> dfdl:valueLength()

3.19. *Sections 7.7*. Additions and clarifications for the defineVariable annotation.

A defaultValue expression must be evaluated before processing the data stream. It is a schema definition error otherwise.

A defaultValue expression can refer to other variables but not to the infoset (so no path locations).The referenced variable must either have a defaultValue or be external. It is a schema definition error otherwise.

If a defaultValue expression references another variable then that prevents the referenced variable's value from ever changing, that is, it is considered to be a read of the variable's value.

If a defaultValue expression references another variable and this causes a circular reference, it is a schema definition error.

If the type of variable is a user-defined simple type restriction, it is a schema definition error.

3.20. *Sections 7.8*. Additions and clarifications for the newVariableInstance annotation.

Only allowed as an annotation on sequence, choice or group reference. It is a schema definition error otherwise.

The resolved set of annotations for a component may contain multiple newVariableInstance statements. They must all be for unique variables, it is a schema definition error otherwise. However, the order of execution among them is not specified. Schema authors can insert sequences to control the timing of evaluation of statements more precisely.

3.21. *Sections 7.9*. Additions and clarifications for the setVariable annotation.

Not allowed as an annotation on a complex element or element reference to such.

The resolved set of annotations for a component may contain multiple setVariable statements. They must all be for unique variables, it is a schema definition error otherwise. However, the order of execution among them is not specified. Schema authors can insert sequences to control the timing of evaluation of statements more precisely.

Clarify that setVariable may be used with a variable defined with external 'true'.

3.22. *New appendix*. Add an explanation of the rationale behind the current variables design, covering why DFDL has adopted a write-once read-many behaviour for variables.

3.23. *Sections 7.3.1*. Additions and clarifications for the assert annotation.

Asserts can be placed as annotations on sequence, choice, group references, local and global element declarations, element references, and simple type definitions.

Replace "More than one dfdl:assert may be used at an annotation point. The dfdl:asserts will be evaluated in the order defined in the schema." with "If the resolved set of annotations for a schema component contain multiple dfdl:assert statements, then those with testKind='pattern' are executed before those with testKind='expression' (the default). However, within each group the order of execution among them is not specified. Schema authors can insert sequences to control the timing of evaluation of statements more precisely."

Once any assert used at an annotation point is unsuccessful, no other asserts are executed at that annotation point.

3.24. *Sections 7.3.1*. Additions and clarifications for the discriminator annotation.

Discriminators can be placed as annotations on sequence, choice, group references, local and global element declarations, element references, and simple type definitions.

Replace "Any one annotation point can contain only a single dfd:discriminator or one or more dfdl:asserts, but not both. It is a schema definition error otherwise." with "The resolved set of annotations for a schema component can contain only a single dfd:discriminator or one or more dfdl:asserts, but not both. It is a schema definition error otherwise."

3.25. *Section 9*. Evaluation Order for Statement Annotations

Of the resolved set of annotations for a schema component, some are statement annotations and the order of their evaluation relative to the actual processing of the schema component itself (parsing or unparsing per its format annotation) is as given in the ordered lists below.

For elements and element refs:
1. dfdl:discriminator or dfdl:assert(s) with testKind='pattern' (parsing only)
2. dfdl:element following property scoping rules
3. dfdl:setVariable(s)
4. dfdl:discriminator or dfdl:assert(s) with testKind='expression' (parsing only)

For sequences, choices and group refs:
1. dfdl:discriminator or dfdl:assert(s) with testKind='pattern' (parsing only)
2. dfdl:newVariableInstance(s)
3. dfdl:setVariable(s)
4. dfdl:sequence or dfdl:choice or dfdl:group following property scoping rules
5. dfdl:discriminator or dfdl:assert(s) with testKind='expression' (parsing only)

<u>Asserts and Discriminators with testKind 'expression'</u>

Implementations are free to optimize by recognizing and executing discriminators or asserts with testKind 'expression' earlier so long as the resulting behavior is consistent with what results from the description above.

<u>Discriminators with testKind 'expression'</u>

When parsing, an attempt to evaluate a discriminator must be made even if preceding statements or the parse of the schema component ended in a processing error.

This is because a discriminator's expression could evaluate to true thereby resolving a point of uncertainty even if the complete parsing of the construct ultimately caused a processing error.

Such discriminator evaluation has access to the DFDL Infoset of the attempted parse as it existed immediately before detecting the parse failure. Attempts to reference parts of the DFDL Infoset that do not exist are processing errors.

Elements and setVariable

The resolved set of dfdl:setVariable statements for an element are executed **after** the parsing of the element. This is in contrast to the resolved set of dfdl:setVariable statements for a group which are executed **before** the parsing of the group.

For elements, this implies that these variables are set after the evaluation of expressions corresponding to any computed DFDL properties for that element, and so the variables may not be referenced from expressions that compute these DFDL properties.

That is, if an expression is used to provide the value of a property (such as dfdl:terminator, or dfdl:byteOrder), the evaluation of that property expression occurs before any dfdl:setVariable annotation from the resolved set of annotations for that element are executed; hence, the expression providing the value of the property may not reference the variable. Schema authors can insert sequences to provide more precise control over when variables are set.

3.26. *Sections 9, 13.15 and others.* Empty, Missing and Defaults.

As originally specified, default values are used as follows. During unparsing, an Infoset with missing required element occurrences is augmented with values so that the resultant data stream that is generated is correct according to the schema and may be successfully re-parsed. During parsing, a sparse data stream with missing required element occurrences has values added to the Infoset so that the resultant Infoset is correct according to the schema.

The parsing behaviour has the effect of making an invalid data stream valid. This is not actually a good idea. Why is DFDL trying to handle missing required occurrences in a data stream? If an occurrence may be missing from the data stream, it should be modelled as optional. Further this is not how XML Schema uses default values for elements.

For elements, XML Schema uses defaults to fill in values for occurrences that are *present but have empty content*. We shall use this principle for DFDL, as the main use case for using defaults on parsing is supplying a value for an empty required occurrence of a simple element (the CSV adjacent separator example). In order for this to work, we must be able to distinguish clearly between an empty occurrence and a missing occurrence when parsing.

Accordingly, formal definitions for *nil representation*, *empty representation*, *normal representation*, and *absent representation* are added to the specification, along with the rules that the parser must use to establish these representations. This is reflected into the grammar. The concept of *missing* from the data stream is redefined. When and how default values are applied when parsing and unparsing are provided.

This is covered in a separate DFDL experience document [DFDLX2].

3.27. *Section 13.6*. Text number rounding.

The DFDL specification behaviour for the properties that control text number rounding was derived from the documented behaviour for ICU4J. However the documentation is not correct. The text number rounding properties are revised as follows to reflect the actual ICU behaviour. In particular note that the way to switch off rounding is to use textNumberRounding ‘explicit’ and new textNumberRoundingMode ‘roundUnnecessary’.

textNumberRounding	Enum Specifies how rounding is controlled during unparsing. Valid values ‘pattern', 'explicit' When dfdl:textNumberRep is 'standard' this property only applies when dfdl:textStandardBase is 10. If 'pattern' then rounding takes place according to the pattern. A rounding increment may be specified in the dfdl:textNumberPattern using digits '1' though '9', otherwise rounding is to the width of the pattern. The rounding mode is always 'roundHalfEven'. If 'explicit' then the rounding increment is specified by the dfdl:textNumberRoundingIncrement property, and any digits '1' through '9' in the dfdl:textNumberPattern are treated as digit '0'. The rounding mode is specified by the dfdl:textRoundingMode property. To disable rounding, use 'explicit' in conjunction with 'roundUnnecessary' for the dfdl:textNumberRoundingMode. If rounding is disabled then any excess precision is treated as a processing error. Annotation: dfdl:element, dfdl:simpleType
textNumberRoundingMode	Enum Specifies how rounding occurs during unparsing, when dfdl:textNumberRounding is 'explicit'. When dfdl:textNumberRep is 'standard' this property only applies when dfdl:textStandardBase is 10. To switch off rounding, use 'roundUnnecessary'. Valid values ‘roundCeiling’, ‘roundFloor’, ‘roundDown’, ‘roundUp’, ‘roundHalfEven’, ‘roundHalfDown’, ‘roundHalfUp', 'roundUnnecessary' Annotation: dfdl:element, dfdl:simpleType
textNumberRoundingIncrement	Double Specifies the rounding increment to use during unparsing, when dfdl:textNumberRounding is 'explicit'. When dfdl:textNumberRep is 'standard' this property only applies when dfdl:textStandardBase is 10. A negative value is a schema definition error. Annotation: dfdl:element, dfdl:simpleType

3.28. *Section 14.3*. Unordered sequence groups.

Replace the wording in the existing section with the following, which clarifies the restrictions on unordered groups and corrects the conceptual rewrite to a repeating choice:

14.3 Unordered Sequence Groups

The occurrences of members of a sequence group with dfdl:sequenceKind='unordered' (hereafter referred to as an 'unordered group') may appear in the data in any order. Occurrences of the same member do not have to be contiguous. In the infoset, sequence groups are always in schema order, so a DFDL processor must sort the members of an unordered group into schema order when parsing. When unparsing, the infoset must already be in schema order, and the members of the sequence will be output in schema order.

14.3.1 Restrictions for Unordered Groups

It is a schema definition error if any member of the unordered group is not an element declaration or an element reference.
It is a schema definition error if a member of an unordered group is an optional element or an array element and its dfdl:occursCountKind property is not 'parsed'
It is a schema definition error if two or more members of the unordered group have the same name and the same namespace (see post-processing transformation below)

14.3.2 Parsing an Unordered Group

When parsing, the semantics of an unordered group are expressed by way of:

1. a source-to-source transformation of the sequence group definition, and
2. a post-processing transformation of the infoset .

An implementation may use any technique consistent with this semantic.

14.3.2.1 Source-to-source Transformation

The source-to-source transformation turns the declaration of an unordered group into an ordered sequence group that contains a repeating choice. To ensure that the resulting schema is a valid DFDL schema, the choice group is wrapped in an array element.
The unordered group is transformed as follows:

- the dfdl:sequenceKind property of the unordered group is changed to "ordered"
- the content of the unordered group is replaced by a complex element (the 'choice element') with the following properties:
    - XSDL minOccurs="0"
    - XSDL maxOccurs="unbounded"
    - dfd:lengthKind="implicit"
    - dfd:occursCountKind="parsed"
- the content of the choice element's complex type is a choice group with the following properties:
    - dfdl:choiceLengthKind="implicit"
- The members of the unordered group become the members of the choice group, with their declaration order preserved.
- The XSDL minOccurs and maxOccurs properties on each member of the choice group are both set to 1.

Using the following example as an illustration:

```
<xs:sequence dfdl:sequenceKind="unordered" dfdl:separator=",">
  <xs:element name="a" type="xs:string"
              dfdl:initiator="A:" />
  <xs:element name="b" type="xs:int" minOccurs="0"
              dfdl:initiator="B:" />
  <xs:element name="c" type="xs:string" minOccurs="0" maxOccurs="10"
              dfdl:initiator="C:" />
</xs:sequence>
```

The above unordered sequence group is conceptually rewritten into the following ordered sequence group:

```
<xs:sequence dfdl:sequenceKind="ordered" dfdl:separator=",">
  <xs:element name="choiceElement" minOccurs="0" maxOccurs="unbounded"
                  occursCountKind="parsed">
    <xs:complexType>
      <xs:choice dfdl:choiceLengthKind="implicit">
        <xs:element name="a" type="xs:string"
                          dfdl:initiator="A:" />
        <xs:element name="b" type="xs:int"
                          dfdl:initiator="B:" />
        <xs:element name="c" type="xs:string"
                          dfdl:initiator="C:" />
      </xs:choice>
    </xs:complexType>
  </xs:element>
</xs:sequence>
```

Processing then constructs a temporary info set for this ordered sequence group by parsing the data.

If a member element is found to have the empty representation then the parsing of that element must use the original value of XSDL minOccurs. In this example, element "b" has minOccurs="0" and if it is found with the empty representation then it must not be defaulted.

14.3.2.2 Post-processing Transformation

Post-processing consists of the following steps:

1. Sort the temporary infoset to produce the real infoset
2. Check scalar elements and validate

Sort the Temporary Infoset

The temporary infoset is transformed into the infoset conforming to the original unordered group. All members of the temporary infoset having the same name and namespace as the first child of the unordered group are placed first, in the order in which they were parsed. This algorithm repeats for the second child of the unordered group and so on until all members of the temporary infoset have been sorted into the schema declaration order of the original unordered group.

For the example above, the temporary infoset is transformed into the infoset corresponding to:

```
<xs:sequence>
  <xs:element name="a" type="xs:string" />
  <xs:element name="b" type="xs:int" minOccurs="0" />
  <xs:element name="c" type="xs:string" minOccurs="0" maxOccurs="10" />
</xs:sequence>
```

Check Scalar Elements and Validate

For each element in the unordered group having XSDL minOccurs="1" and maxOccurs="1", the number of occurrences is checked. Each such element must occur exactly once in the infoset, else it is a processing error.

If validation is enabled, the DFDL processor validates the number of occurrences of each member of the unordered group against XSDL minOccurs and maxOccurs.

These checks are the same as those performed for an ordered sequence group. However, in an unordered group the checking of XSDL minOccurs and maxOccurs must be performed *after* the entire group has been parsed.

14.3.3 Unparsing an Unordered Group
When unparsing, the behavior is exactly as if dfdl:sequenceKind='ordered'. The infoset must be presented to the unparser in schema declaration order, and the members of the unordered sequence group are output in schema declaration order.

3.29. *Sections 24 and 30*. The DFDL specification is not prescriptive enough when specifying what is allowed for regular expressions used in the length property and testPattern property.

Section 24 is replaced by the following.

"A DFDL regular expression may be specified for the dfdl:lengthPattern format property and the dfdl:testPattern attribute of the dfdl:assert and dfdl:discriminator annotations. DFDL regular expressions do not interpret DFDL entities.

A DFDL regular expression is defined by a set of valid pattern characters. For portability, a DFDL regular expression pattern is restricted to the inclusive subset of the ICU regular expression [ICURE] and the Java(R) 7 regular expression [JAVARE] with the Unicode flags UNICODE_CASE and UNICODE_CHARACTER_CLASS turned on. DFDL regular expressions thereby conform to Unicode Technical Standard #18, Unicode Regular Expressions, level 1 [UNICODERE].

The following regular expression constructs are not common to both ICU and Java(R) 7 and it is a schema definition error if any are used in a DFDL regular expression:

Construct	**Meaning**	**Notes**
\N{UNICODE CHARACTER NAME}	Match the named character	ICU only
\X	Match a Grapheme Cluster	ICU only
\Uhhhhhhhh	Match the character with the hex value hhhhhhhh	ICU only
(?# …)	Free-format comment	ICU only
(?w-w)	UREGEX_UWORD - Controls the behaviour of \b in a pattern	ICU only
(?d-d)	UNIX_LINES - Enables Unix lines mode	Java 7 only
(?u-u)	UNICODE_CASE - Enables Unicode-aware case folding	Java 7 only (1)
(?U-U)	UNICODE_CHARACTER_CLASS - Enables the Unicode version of predefined character classes and POSIX character classes	Java 7 only (2)

Notes:
(1) Implementations using Java 7 must set flag UNICODE_CASE by default to match ICU.
(2) Implementations using Java 7 must set flag UNICODE_CHARACTER_CLASS by default to match ICU.

Additionally, the behaviour of the word character construct (\w) is not consistent in ICU and Java 7. In Java 7 \w is [\p{Alpha}\p{gc=Mn}\p{gc=Me}\p{gc=Mc}\p{Digit}\p{gc=Pc}], which is a larger set than ICU where \w is [\p{Ll}\p{Lu}\p{Lt}\p{Lo}\p{Nd}].
The use of \w is not recommended in DFDL regular expressions in conjunction with Unicode encodings, and an implementation must issue a warning if such usage is detected.

Character properties are detailed by the Unicode Regular Expressions [UNICODERE]. "

Section 30 is updated to correct the references used in section 24:
- *Add:* [ICURE] - http://userguide.icu-project.org/strings/regexp
- *Add:* [UNICODERE] - http://www.unicode.org/reports/tr18/
- *Remove:* [PERLRE] - http://perldoc.perl.org/perlre.html#Extended-Patterns

- *Change:* [JAVARE] - http://docs.oracle.com/javase/7/docs/api/java/util/regex/Pattern.html

3.30. *Section 16.* Changes to placement of occurs properties.

Remove the restriction that DFDL occurs properties are not applicable to global elements. This rule causes problems when applying property scoping rules. DFDL occurs properties may now be specified on global elements.

Also see update to errata 3.8.

3.31. *Section 14.5.* Clarifications to hidden groups.

When unparsing a hidden group, the behaviour should be the same as when elements are missing from the infoset; that is, the default values algorithm applies. The only difference is that if a required element does not have a default value or a dfdl:outputValueCalc then it is a schema definition error instead of a processing error.

When unparsing a hidden group, it is a processing error if an element information item is provided in the infoset for an element contained within the bounds of a hidden group.

4. Revised Grammar

This chapter provides a consolidated grammar that incorporates the several errata in this document that affect it.

Productions
Document = ***UnicodeByteOrderMark*** DocumentElement
DocumentElement = SimpleElement | ComplexElement
SimpleElement = SimpleLiteralNilElementRep | SimpleEmptyElementRep | SimpleNormalRep
SimpleEnclosedElement = SimpleElement | AbsentElementRep
ComplexElement = ComplexLiteralNilElementRep | ComplexNormalRep | ComplexEmptyElementRep
ComplexEnclosedElement = ComplexElement | AbsentElementRep
EnclosedElement = SimpleEnclosedElement | ComplexEnclosedElement
AbsentElementRep = ***Absent***
SimpleEmptyElementRep = EmptyElementLeftFraming EmptyElementRightFraming
ComplexEmptyElementRep = EmptyElementLeftFraming EmptyElementRightFraming
EmptyElementLeftFraming = LeadingAlignment ***EmptyElementInitiator*** PrefixLength
EmptyElementRightFraming = ***EmptyElementTerminator*** TrailingAlignment
SimpleLiteralNilElementRep = NilElementLeftFraming [***NilLiteralCharacters*** | NilElementLiteralContent] NilElementRightFraming
ComplexLiteralNilElementRep = NilElementLeftFraming ***NilLiteralValue*** NilElementRightFraming
NilElementLeftFraming = LeadingAlignment ***NilElementInitiator*** PrefixLength
NilElementRightFraming = ***NilElementTerminator*** TrailingAlignment
NilElementLiteralContent = ***LeftPadding*** ***NilLiteralValue*** RightPadOrFill
SimpleNormalRep = LeftFraming PrefixLength SimpleContent RightFraming
ComplexNormalRep = LeftFraming PrefixLength ComplexContent ***ElementUnused*** RightFraming
LeftFraming = LeadingAlignment ***Initiator***
RightFraming = ***Terminator*** TrailingAlignment
PrefixLength = SimpleContent | PrefixPrefixLength SimpleContent
PrefixPrefixLength = SimpleContent
SimpleContent = ***LeftPadding*** [***NilLogicalValue | SimpleValue***] RightPadOrFill
ComplexContent = Sequence | Choice

Sequence = LeftFraming SequenceContent RightFraming
SequenceContent = [***PrefixSeparator*** EnclosedContent [***Separator*** EnclosedContent]* ***PostfixSeparator***]
 	
Choice = LeftFraming ChoiceContent RightFraming
ChoiceContent = [EnclosedContent] ***ChoiceUnused***
 	
EnclosedContent = [EnclosedElement | Array | Sequence | Choice]
 	
Array = [EnclosedElement [***Separator*** EnclosedElement]* [***Separator*** StopValue]]
StopValue = SimpleElement

LeadingAlignment = ***LeadingSkip AlignmentFill***
TrailingAlignment = ***TrailingSkip***
RightPadOrFill = ***RightPadding*** | ***RightFill* | *RightPadding RightFill***

5. Security Considerations

Security considerations are dealt with in the corresponding sections of the DFDL 1.0 specification [DFDL].

No additional security issues have been raised.

6. Contributors

Stephen M. Hanson,
IBM Software Group,
Hursley,
Winchester,UK
smh@uk.ibm.com

Michael J. Beckerle,
Tresys Technology,
Columbia, MD, USA
mbeckerle@tresys.com

We greatly acknowledge the contributions made to this document by the following people.

Tim Kimber, IBM Software Group, Hursley, UK
Stephanie Fetzer, IBM Software Group, Charlotte, USA
Richard Schofield, IBM Software Group, Hursley, UK
Suman Kalia, IBM Software Group, Markham, Toronto, Canada
Jonathan Cranford, Mitre Corporation, USA

7. Intellectual Property Statement

8. Disclaimer

9. Full Copyright Notice

10. References

[DFDL] OGF DFDL 1.0 specification
http://www.ogf.org/documents/GFD.174.pdf/

[DFDLR] OGF DFDL 1.0 specification - revised
<To be added>

Comment [SMH2]: Complete reference when GFD number allocated.

[GFD] OGF Document Process and Requirements
http://www.ogf.org/documents/GFD.152.pdf/

[ULDML] UTS #35: Unicode Locale Data Markup Language (LDML)
http://www.unicode.org/reports/tr35/

[UCLDR] Unicode Common Locale Data Repository
https://sites.google.com/site/cldr/

[DFDLX2] DFDL Experience Document 2
<To be added>

Comment [SMH3]: Complete reference when GFD number allocated.

[XSDL2] XML Schema Part 2: Datatypes Second Edition
http://www.w3.org/TR/2004/REC-xmlschema-2-20041028/

[ICURE] ICU Regular Expressions
http://userguide.icu-project.org/strings/regexp

[UNICODERE] Unicode Regular Expressions
http://www.unicode.org/reports/tr18/

[JAVARE] Java 7 Regular Expressions
http://docs.oracle.com/javase/7/docs/api/java/util/regex/Pattern.html

[XPATH2] XPath 2.0
http://www.w3.org/TR/xpath20/