Michael J Beckerle

Stephen M Hanson

Alan W Powell

March 2013

Data Format Description Language (DFDL) v1.0 Specification

NOTE: THIS IS AN INTERNAL DFDL WORKING GROUP DRAFT DOCUMENT. EVERYTHING IN HERE IS STILL SUBJECT TO CHANGE.

Status of This Document

This document provides information to the OGF community on a standard Data

Format Description Language (DFDL). Distribution is unlimited.

This is a working draft that is in the process of being updated to incorporate identified errata.

All changes due to errata are flagged as Word comments.

Abstract

This document provides a definition of a standard Data Format Description Language (DFDL). This language allows description of dense binary and legacy data formats in a vendor-neutral declarative manner. DFDL is an extension to the XML Schema Description Language (XSDL).

Table of contents

1. Introduction

Data interchange is critically important for most computing. Grid computing and all forms of distributed computing require distributed software and hardware resources to work together. Inevitably, these resources read and write data in a variety of formats. General tools for data interchange are essential to solving such problems. Scalable and High Performance Computing (HPC) applications require high-performance data handling, so data interchange standards must enable efficient representation of data. Data Format Description Language (DFDL) enables powerful data interchange and very high-performance data handling.

We envisage three dominant kinds of data in the future, as follows:

1. Textual data defined by a format specific schema such as XML or JSON.

2. Binary data in standard formats.

3. Data with DFDL descriptors.

Textual XML data is the most successful data interchange standard to date. All such data are by definition new, by which we mean created in the XML era. Because of the large overhead that XML tagging imposes, there is often a need to compress and decompress XML data. However, there is a high-cost for compression and decompression that is unacceptable to some applications. Standardized binary data are also relatively new, and is suitable for larger data because of the reduced costs of encoding and more compact size. Examples of standard binary formats are data described by modern versions of ASN.1, or the use of XDR. These techniques lack the self-describing nature of XML-data. Scientific formats, such as NetCDF and HDF are used by some communities to provide self-describing binary data. There are also standardized binary-encoded XML data formats such as EXI..

It is an important observation that both XML format and standardized binary formats are *prescriptive* in that they specify or prescribe a representation of the data. To use them your applications must be written to conform to their encodings and mechanisms of expression.

DFDL suggests an entirely different scheme. The approach is *descriptive* in that one chooses an appropriate data representation for an application based on its needs and one then describes the format using DFDL so that multiple programs can directly interchange the described data. DFDL descriptions can be provided by the creator of the format, or developed as needed by third parties intending to use the format. That is, DFDL is not a format for data; it is a way of describing any data format. DFDL is intended for data commonly found in scientific and numeric computations, as well as record-oriented representations found in commercial data processing.

DFDL can be used to describe legacy data files, to simplify transfer of data across domains without requiring global standard formats, or to allow third-party tools to easily access multiple formats. DFDL can also be a powerful tool for supporting backward compatibility as formats evolve.

DFDL is designed to provide flexibility and also permit implementations that achieve very high levels of performance. DFDL descriptions are separable and native applications do not need to use DFDL libraries to parse their data formats. DFDL parsers can also be highly efficient. The DFDL language is designed to permit implementations that use lazy evaluation of formats and to support seekable, random access to data. The following goals can be achieved by DFDL implementations:

• Density. Fewest bytes to represent information (without resorting to compression). Fastest possible I/O.

• Optimized I/O. Applications can write data aligned to byte, word, or even page boundaries and to use memory-mapped I/O to insure access to data with the smallest number of machine cycles for common use cases without sacrificing general access.

DFDL can describe the same types of abstract data that other binary or textual data formats can describe and, furthermore, it can describe almost any possible representation scheme for those data. It is the spirit of DFDL to support canonical data descriptions that correspond closely to the original in-memory representation of the data, and also to provide sufficient information to write as well as to read the given format.

1.1   Why is DFDL Needed?

The question arises of why DFDL is needed in an era when there are so many standard data formats available. Ultimately, it is because there are a number of social phenomena in the way software is developed that have lead to the situation today where DFDL is needed to standardize descriptions of diverse data formats.

First, programs are very often written speculatively, that is, without any advance understanding of how important they will become. Given this situation, little effort is expended on data formats since it remains easier to program the I/O in the most straightforward way possible with the programming tools in use. Even something as simple as using an XML-based data format is harder than just using the native I/O libraries of a programming language.

In time, however, it is realized that a software program is important because either many people are using it, or it has become important for business or organizational needs to start using it in larger scale deployments. At that point it is often too late to go back and change the data formats. For example, there may be real or perceived business costs to delaying the deployment of a program for a rewrite just to change the data formats, particularly if such rewriting will reduce the performance of the program and increase the costs of deployment. (It takes *longer* to program, but at least it's *slower* when you are done☺)

Additionally, the need for data format standardization for interchange with other software may not be clear at the point where a program first becomes 'important'. Eventually, however, the need for data interchange with the program becomes apparent.

The above phenomena are not something that is going away any time soon. There are, of course, efforts to smoothly integrate standardized data format handling into programming languages. Nevertheless, we see a critical role for DFDL since it allows after-the-fact description of a data format.

1.2   What is DFDL?

DFDL is a language for describing data formats. A DFDL description allows data to be read from its native format and to be presented as an instance of an information set or indeed converted to the corresponding XML document. DFDL also allows data to be taken from an instance of an information set and written out to its native format.

DFDL achieves this by leveraging W3C XML Schema Definition Language (XSDL) 1.0. [XSDLV1]

An XML schema is written for the logical model of the data. The schema is augmented with special DFDL annotations. These annotations are used to describe the native representation of the data. This is an established approach that is already being used today in commercial systems.

1.2.1   Simple Example

Consider the following XML data:

```
<w>5</w>
<x>7839372</x>
<y>8.6E-200</y>
<z>-7.1E8</z>
```

The logical model for this data can be described by the following fragment of an XML schema document that simply provides description of the name and type of each element:

```
<xs:complexType name="example1">
  <xs:sequence>
    <xs:element name="w" type="xs:int"/>
    <xs:element name="x" type="xs:int"/>
    <xs:element name="y" type="xs:double"/>
    <xs:element name="z" type="xs:float"/>
  </xs:sequence>
</xs:complexType>
```

Now, suppose we have the same data but represented in a non-XML format. A binary representation of the data could be visualized like this (shown as hexadecimal):

```
0000 0005 0077 9e8c

169a 54dd 0a1b 4a3f

ce29 46f6
```

To describe this in DFDL, we take our original XML schema document that described the data model and we annotate the type definition as follows:

```
<xs:complexType >
  <xs:sequence dfdl:byteOrder="bigEndian">
    <xs:element name="w" type="xs:int">
      <xs:annotation>
        <xs:appinfo source="http://www.ogf.org/dfdl/">
          <dfdl:element representation="binary"
                    binaryNumberRep="binary"
                    byteOrder="bigEndian"
                    lengthKind="implicit"/>
        </xs:appinfo>
      </xs:annotation>
    </xs:element>
    <xs:element name="x" type="xs:int ">
      <xs:annotation>
        <xs:appinfo source="http://www.ogf.org/dfdl/">
          <dfdl:element representation="binary"
                    binaryNumberRep="binary"
                    byteOrder="bigEndian"
                    lengthKind="implicit"/>
        </xs:appinfo>
      </xs:annotation>
    </xs:element>
    <xs:element name="y" type="xs:double">
      <xs:annotation>
        <xs:appinfo source="http://www.ogf.org/dfdl/">
          <dfdl:element representation="binary"
                    binaryFloatRep="ieee"
                    byteOrder="bigEndian"
                    lengthKind="implicit"/>
        </xs:appinfo>
      </xs:annotation>
    </xs:element>
    <xs:element name="z" type="xs:float" >
      <xs:annotation>
        <xs:appinfo source="http://www.ogf.org/dfdl/">
          <dfdl:element representation="binary"
                    byteOrder="bigEndian"
                    lengthKind="implicit"
```

```
                    binaryFloatRep="ieee" />
          </xs:appinfo>
        </xs:annotation>
      </xs:element>
    </xs:sequence>
  </xs:complexType>
```

This simple DFDL annotation expresses that the data are represented in a binary format and that the byte order will be big endian. This is all that a DFDL parser needs to read the data.

Consider if the same data are represented in a text format:

```
5,7839372,8.6E-200,-7.1E8
```

Once again, we can annotate the same data model, this time with properties that provide the character encoding, the field separator (comma) and the decimal separator (period):

```
  <xs:complexType >
    <xs:sequence >
      <xs:annotation>
        <xs:appinfo source="http://www.ogf.org/dfdl/">
          <dfdl:sequence encoding="UTF-8" byteOrder="bigEndian"
                         separator="," />
        </xs:appinfo>
      </xs:annotation>
      <xs:element name="w" type="xs:int">
        <xs:annotation>
          <xs:appinfo source="http://www.ogf.org/dfdl/">
            <dfdl:element representation="text"
                        encoding="UTF-8"
                        textNumberRep ="standard"
                         textNumberPattern="####0"
                        textStandardGroupingSeparator=","
                        textStandardDecimalSeparator="."
                        lengthKind="delimited"/>
          </xs:appinfo>
        </xs:annotation>
      </xs:element>
      <xs:element name="x" type="xs:int">
        <xs:annotation>
          <xs:appinfo source="http://www.ogf.org/dfdl/">
            <dfdl:element representation="text"
                        encoding="UTF-8"
                        textNumberRep ="standard"
                         textNumberPattern="######0"
                        textStandardGroupingSeparator=","
                        textStandardDecimalSeparator="."
                        lengthKind="delimited"/>
          </xs:appinfo>
        </xs:annotation>
      </xs:element>
      <xs:element name="y" type="xs:double">
        <xs:annotation>
          <xs:appinfo source="http://www.ogf.org/dfdl/">
            <dfdl:element representation="text"
                        encoding="UTF-8"
                        textNumberRep ="standard"
                         textNumberPattern="0.0E+000"
```

```
                        textStandardGroupingSeparator=","
                        textStandardDecimalSeparator="."
                        lengthKind="delimited"/>
        </xs:appinfo>
      </xs:annotation>
    </xs:element>
    <xs:element name="z" type="xs:float">
      <xs:annotation>
        <xs:appinfo source="http://www.ogf.org/dfdl/">
          <dfdl:element representation="text"
                        encoding="UTF-8"
                        textNumberRep ="standard"
                         textNumberPattern="0.0E0"
                        textStandardGroupingSeparator=","
                        textStandardDecimalSeparator="."
                        lengthKind="delimited"/>
        </xs:appinfo>
      </xs:annotation>
    </xs:element>
  </xs:sequence>
</xs:complexType>
```

1.3   What DFDL is not

DFDL maps data from a non-XML representation to an instance of an information set. This can be thought of as a data transformation. However, DFDL is not intended to be a general transformation language and, in particular, DFDL does not intend to provide a mechanism to map data to arbitrary XML models. There are two specific limitations on the data models that DFDL can work to:

1. DFDL uses a subset of XML Schema, in particular, you cannot use XML attributes in the data model.

2. The order of the data in the data model must correspond to the order and structure of the data being described.

This latter point deserves some elaboration. The XML schema used must be suitable for describing the physical data format. There must be a correspondence between the XML schema's constructs and the physical data structures. For example, generally the elements in the XML schema must match the order of the physical data. DFDL does allow for certain physically unordered formats as well.

The key concept here is that when using DFDL, you do not get to design an XML schema to your preference and then populate it from data. That would involve describing the data format, and describing a transformation for mapping it to the XML schema you have designed. DFDL is only about the format part of this problem. There are other languages, such as XSLT, which are for transformation. In DFDL, you describe only the format of the data, and this format constrains the nature of the XML schema you must use in its description.

1.4   Scope of version 1.0

The goals of version 1.0 are as follows:

1. Leverage XML technology and concepts

2. Support very efficient parsers/formatters

3. Avoid features that require unnecessary data copying

4. Support round-tripping, that is, read and write data in a described format from the same description

5. Keep simple cases simple

6. Simple descriptions should be "human readable" to the same degree that XSDL is.

The general features of version 1.0 are as follows:

a) Text and binary data parsing and unparsing

b) Validate the data when parsing and unparsing using XSDL validation.

c) Defaulted input and output for missing representations

d) Reference – use of the value of a previously read element in subsequent expressions

e) Choice – capability to select among format variations

f) Hidden sequence of elements – A description of an intermediate representation whose corresponding Infoset is not exposed in the final result.

g) Basic Math – in DFDL expressions

h) Out-of-type value handling (e.g., The string value 'NIL' to indicate nil for an integer)

i) Speculative parsing to resolve uncertainty.

j) Very general parsing capability: Lookahead/Push-back

Version 1.0 of DFDL is a language capable of expressing a wide range of binary and text-based data formats.

DFDL is capable of describing binary data as found in the data structures of COBOL, C, PL1, Fortran, etc. In particular, it is able to describe repeating sub-arrays where the length of an array is stored in another location of the structure.

DFDL is capable of describing a wide variety of textual data formats such as HL7, X12, and SWIFT. Textual data formats often use syntax delimiters, such as initiators, separators and terminators to delimit fields.

DFDL has certain composition properties. I.e., two formats can be nested or concatenated and a working format results.

The following topics have been deferred to future versions of the standard:

- Extensibility: There are real examples of proprietary data format description languages that we use as our base of experience from which to derive standard DFDL. However, there are no examples of extensible format description languages. Therefore, while extensibility is desirable in DFDL, there is not yet a base of experience with extensibility from which to derive a standard.

- Rich Layering: Some formats require data to be described in multiple passes. Combining these into one DFDL schema requires very rich layering functionality. In these layers one element's value becomes the representation of another element. DFDL V1.0 allows description of only a limited kind of layering.

1.5   Related standards

1. Prescriptive systems:

   a. JSON

   b. EXI (binary XML) (http://www.w3.org/TR/exi)

   c. Thrift (http://thrift.apache.org/static/files/thrift-20070401.pdf)

   d. Avro (http://avro.apache.org/docs/1.3.0/spec.html)

   e. ASN.1 with any of the prescribed encoding rules: Basic Encoding Rules (BER),
       Distinguished Encoding Rules (DER), Canonical Encoding Rules(CER)
       (http://www.itu.int/rec/T-REC-X.690-200811-I/en) or Packed Encoding Rules
       (PER) (http://www.itu.int/rec/T-REC-X.691-200811-I/en)

2. Descriptive systems:

   a. ASN1 Encoding Control Notation (also known as ITU-T X.692)
       (http://www.itu.int/rec/T-REC-X.692-200811-I)

Comment [SMH1]: Add GPB

Comment [m2]: (DONE) Is GPB really a 'related' 'standard'. It's not a standard in any official sense, just defacto used by google (and others), and it's just a binary wire format thing with a specific representation. I don't think we should get into the enumeration of binary format "quasi-standards" here. There's a good Wikipedia page about comparison of data formats.

DONE

Comment [m3]: Changed "future" language to present tense and citation to EXI which is a w3c recommendation.

Comment [SMH4]: I would have said ASN.1 is prescriptive – it is self-describing but it forces you to use one of several syntaxes.

Comment [m5]: Clarified ASN.1, which has prescriptive encodings and a descriptive ECN thing.

2. Notational and Definitional Conventions

The key words *must*, *must not*, *required*, *shall*, *shall not*, *should*, *should not*, *recommended*, *may*, *may not* and *optional* in this document are to be interpreted as described in [RFC 2119]. Note that for reasons of clarity these words are not always capitalized in this document.

Examples are for illustration purposes only and for clarity they will often not include all the necessary DFDL properties.

2.1 Failure Types

Where the phrase "must be consistent with" is used, it is assumed that a conforming DFDL implementation must check for the consistency and issue appropriate diagnostic messages when an inconsistency is found.

There are several kinds of failures that can occur when a DFDL processor is handling data and/or a DFDL schema.

2.2 Schema Definition Error

When the DFDL schema itself contains an error, it implies that the DFDL processor cannot process data because the DFDL schema is not meaningful. It may be ambiguous, or contain conflicting definitions. Equivalently, we can say that there is no possible data that conforms to the schema; hence, the schema cannot be meaningful. All conforming DFDL processors must detect all schema definition errors, and must issue some kind of appropriate diagnostic message. The behavior of a DFDL processor after a schema definition error is detected is out of scope for this specification.

When a Schema definition error can be detected given only the schema, it is desirable, though not required by the DFDL standard, that such errors be detected and diagnostic messages issued before any data are processed. Of course not all schema definition errors can be detected without reference to data as some representation properties may obtain their values from the data (see section 2.3.1 Ambiguity of Data Formats).

The expression language included within DFDL is strongly, statically type checkable. This means that type checking of expressions can be performed without processing data, and implementations are encouraged to perform this checking statically so that schema definition errors having to do with type inconsistencies can be detected before processing data.

Note that schema definition errors cannot be suppressed by points of uncertainty.

In addition, a DFDL processor:

> o That only implements a DFDL parser does not have to check (for schema definition errors) properties that are solely used when unparsing, though it is recommended that it does so for portability reasons.

> o That does not implement some optional features does not have to check properties or annotations required by those optional features, but MUST issue a warning that an unrecognized property or annotation has been encountered.

> o Need not check global objects as they may legitimately be incomplete due to properties intended to be supplied based on scoping rules and the context at the point of use,

> There are two exceptions to this, which must be checked:

>> o Global simple types that are referenced by prefixLengthType property

>> o Global elements that are the document root.

Some situations suggest likely errors, but a DFDL processor cannot be certain. In these situations, a DFDL processor may issue warnings to assist a DFDL schema author in identifying likely errors. An important case of this is when the DFDL processor encounters a schema

Comment [SMH6]: Errata 3.8 affects this section

DONE

Comment [SMH7]: Update to 3.8. DONE

component and annotation where there are explicitly properties that are not relevant to the component as defined. Depending on the specifics of the component and property the DFDL processor can or must take certain actions. If the:

o   Property is not applicable to the component's DFDL annotation.

    Schema definition error. Example is lengthKind on xs:sequence.

o   Property is not applicable because of simple type.

    Warning (optional). Example is calendarPatternKind on xs:string.

o   Property is not applicable because of another DFDL property setting.

    Warning (optional). Example is binaryNumberRep when representation is text.

o   Property is not applicable because object is local, global or reference.

    Warning (optional). Example is occursCountKind on a global xs:element.[1]

2.2.1.1 Schema Component Constraint: Unique Particle Attribution

> **Comment [m8]:** Moved here since UPA is a schema definition error.

A DFDL processor MUST implement the Schema Component Constraint: Unique Particle Attribution defined in *XML Schema Part 1: Structures* [XSDLV1] that applies to the DFDL schema subset.

Two elements **overlap** if

- They are both element declaration particles whose declarations have the same name and target namespace.

A schema will violate the unique attribution constraint if it contains two particles which overlap and which either

- Are both in the particles of a *choice* group

Or

- Either may validate adjacent information items and the first has XSDL minOccurs less than XSDL maxOccurs.

2.3   Processing Errors

If a DFDL schema contains no schema definition errors, then there is the additional possibility that when processing data using a DFDL schema, the data do not conform to the format described by the schema. This is known as a *processing error*.
Processing errors can be suppressed by a point of uncertainty. See section 1.1

> **Comment [SMH9]:** Affected by action 140.

> **Comment [m10]:** DONE. Kept out of the detals here, but now consistent with other 140 materials.
>
> *SMH: Removed ref to dfdl:occursCountKind as DFDL properties have not yet been introduced*

It is expected that DFDL implementations will provide additional mechanisms for dealing with effective processing errors such as the means of specifying retry points or the means of skipping some data so as to recover from the error in some way.

Exceptions that occur in the evaluation of the DFDL expression language are processing errors.

Non-conformance with the XSDL *minOccurs* or *maxOccurs* constraints is either a processing error or only a validation error depending on the settings of certain DFDL properties (see section 16 below).

> **Comment [m11]:** DONE
>
> Do we need to provide more details here? About the fact that for OCK='implicit' we use the maxOccurs to drive the parse.
>
> We could say…
>
> However, some DFDL parser behaviors will only parse a number of occurrences less than or equal to maxOccurs; hence, this validation error will not be possible in those cases.
>
> *SMH: When unparsing exceeding maxOccurs can be a processing error, so combined the two paras. I think that is sufficient for now.*

2.3.1   Ambiguity of Data Formats

---

[1]   Excludes inputValueCalc and outputValueCalc.

A data format using delimiters may be ambiguous if the delimiters are not distinct and a data format description which has fixed data requirements (that is, where some elements have fixed values), may be ambiguous even with fixed length elements.[2]

If the delimiter string values are stored within the data, perhaps as elements of a header part of the data, then this ambiguity certainly cannot be examined until the data is available.

Given an ambiguous grammar, a DFDL implementation may successfully parse a particular input data stream. That is, the part of the schema with the ambiguity may not be exercised by a particular data stream, or the data may parse successfully anyway because the ambiguity may not cause any kind of failure or processing error.

Hence, to insure compatible behavior, DFDL v1.0 implementations MUST NOT detect grammar ambiguities as errors. Implementations are of course free to issue warnings to help users identify these situations, but ambiguity is neither a Schema Definition Error nor a Processing Error.

2.3.1.1 Unparsing Must be Unambiguous

Usually, the behavior of the unparser is symmetric to the behavior of the parser; however, there are cases where the DFDL schema will accept several equivalent representations for the same logical data. In this case it would be ambiguous which of these equivalent representations should be produced by the unparser. The DFDL standard contains representation properties which are used to eliminate this ambiguity. It is a schema definition error if a DFDL schema is being used to unparse data and there is any ambiguity about the representation.

2.4   Validation Errors

Logical validation checks are constraints expressed in XSDL and they apply to the logical value of the infoset. Hence, parsing must successfully construct the infoset from the representation in order for validation checks to be meaningful. This implies that validation errors cannot affect the ability of a DFDL processor to successfully parse or unparse data.

DFDL processors may provide both validating and non-validating behaviors on either or both of parse and unparse. (A DFDL implementation could support validate on parse, but not support it on unparse and still be considered conforming.)

The behavior of a DFDL processor after a validation error is not specified by the DFDL language.

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.

When resolving points of uncertainty, validation errors are ignored.

Comment [SMH12]: Errata 3.12 changes this

DONE

The following DFDL schema constructs are allowed in DFDL and are checked when validating:

1. XSDL pattern facet - (for xs:string type elements only)

2. XSDL minLength, maxLength

3. XSDL minInclusive, minExclusive, maxInclusive, maxExclusive

4. XSDL enumeration

5. XSDL maxOccurs

---

[2] A very complex analysis is needed to identify this sort of grammar ambiguity in general. While we believe this may be decidable for DFDL v1.0, future versions of DFDL may add features (such as recursive types) which make this analysis undecidable.

Note that validation is distinct from the checking of DFDL assert or discriminator predicates. When a DFDL discriminator or assert is used to discriminate a choice or other point of uncertainty when parsing, then that assert or discriminator is essential to parsing and it is evaluated irrespective of whether validation is enabled or disabled.

Note that validation errors cannot be suppressed by points of uncertainty.

2.5   Recoverable Error

This error type is used with the dfdl:assert annotation when parsing to permit the checking of physical format constraints without terminating a parse. For example, some formats will have redundancy by having known lengths, as well as delimiters. A recoverable error can be issued, using an assert to check a physical length constraint when property lengthKind is 'delimited'.

Recoverable errors are independent of validation, and when resolving points of uncertainty, recoverable errors are ignored.

> **Comment [SMH13]:** Errata 3.4 adds a new failure type called 'Recoverable error'. Also affects section 7.3.
>
> DONE

3. Glossary

Adjacent - Two parts of the input/output stream are adjacent if they are at consecutive addresses.

Addressable Unit, or Unit - This is the unit of storage that makes up the input or output stream holding the representation of the data. The units are bits, bytes, or characters.

Annotation point - A location within a DFDL schema where DFDL annotation elements are allowed to appear.

Applicable properties - All the DFDL properties that apply to that type of schema construct. For example all the DFDL properties that apply to an xs:simpleType.

Array - The set of adjacent elements whose XSDL element declaration specifies the potential for it to have more than one occurrence (XSD s:maxOccurs > '1' or 'unbounded'). Of course any given array can have any number of element occurrences, including zero elements or exactly 1 element as long as the occurrence constraints are met. If XSDL maxOccurs is 'unbounded' then there is no constraint to the maximum number of occurrences, though implementations may have maximum capabilities. An optional element (XSDL maxOccurs='1', minOccurs='0') is not considered to be an array as described in this document. Note that a sequence is not to be confused with an array. A sequence is a complex tuple type for an element; the children of a sequence can be of different types. All elements of an array have the same type and have the same information item members except for the value member.

Array Element – an element declaration or reference with xs:maxOccurs > '1' or 'unbounded'.

Augmented Infoset - When unparsing one begins with the DFDL schema and conceptually with the logical infoset. As the values of items are filled in by defaulting, and by use of the DFDL outputValueCalc property (including on hidden items), these new item values augment the infoset. The resulting infoset is called the augmented infoset.

Byte - The term "byte" refers to an 8-bit octet.

Component - A construct within a DFDL schema that may contain a DFDL annotation..

Content - The content is the bits of data that are interpreted to compute a logical value.

Content Model - Used in describing the syntactic structure of XSD and DFDL annotations of it. An element of a schema can have empty, simple, element-only, or mixed content. An element declaration for an element of complex type containing a xs:sequence element is said to have a sequence in its content model.

Contiguous - An element has a contiguous representation if all parts of its representation are adjacent in the input/output stream. Most simple types have contiguous representations naturally. Groups containing elements that are themselves contiguous are also considered to have contiguous representations irrespective of alignment fill or padding of any kind that exists within the group. Similarly, arrays of elements that are themselves contiguous are also contiguous. An example of a non-contiguous representation would be a nillable element, where a flag is used to determine whether or not the element is nil, and the location of that flag is not adjacent to the value representation.

Count - The number of occurrences of an element.

Defining Annotations - The annotation elements dfdl:defineFormat, dfdl:defineVariable, and dfdl:defineEscapeScheme

Delimiter - A character or string used to separate, or mark the start and end of, items of data. In DFDL, dfdl:lengthKind 'delimited' searches for separators and terminators.

Delimiter scanning - When parsing, the process of scanning for a specific item in the data which marks the end of an item, or the beginning of a subsequent item is referred to as delimiter scanning, or simply scanning for short. Scanning also takes into account escape schemes so as to allow the delimiters to appear within data if properly escaped.

Comment [SMH14]: Add that When resolving points of uncertainty, recoverable errors are ignored.

DONE

Comment [m15]: 2.108. Updates to Glossary entries DONE.

Section 3. Use the new or changed terms as appropriate throughout the specification: Annotation point(DONE), Statement annotations,(DONE) defining annotations,(DONE) format annotations,(DONE) resolved set of annotations.(DONE)

Comment [m16]: 2.112. *Section 3*. Update the Glossary concerning arrays, as follows DONE

use the or changed terms as appropriate throughout the specification (DONE – except for the sections that will be replaced when the issue 140 material is inserted.)

o *Remove:* Fixed-Occurrence Item(DONE)
o *Remove:* Variable Occurrence Item (DONE)
o *Remove:* Optional Item (DONE)
o *Remove:* Number Of Occurrences (DONE – from the glossary. We use this phrase in many places in the spec.)
o *Change:* Required Element.(DONE)
o *Change:* Optional Element (DONE)
o *Add:* Fixed Array Element. (DONE)
o *Add:* Variable Array Element (DONE)
o *Add:* Occurrence.(DONE)
o *Add:* Count. (DONE)
o *Add:* Index. (DONE)
o *Add:* Required Occurrence. (DONE)
Add: Optional Occurrence (DONE)

DFDL – Data Format Description Language

DFDL Processor - A program that uses DFDL schemas in order to process data described by them.

DFDL Schema - an XML schema containing DFDL annotations to describe data format.

Dynamic extent - This is a characteristic of the data stream. When parsing a declaration or definition, the collection of bits within the data stream that contain any aspect of the representation of that element make up the element's dynamic extent.

Dynamic scope - This is a characteristic of parts of the DFDL schema. When a definition or declaration contains or references another declaration or definition, then the contained definition or declaration is said to be in the dynamic scope of the enclosing one. The important characteristic of dynamic scoping is that it traverses references. When parsing, the dynamic scope of an element declaration includes all definitions and declarations used as part of parsing that element.

Element - A part of the data described by an element declaration in the schema and presented as an element information item in the infoset.

Explicit properties - The explicit properties are the combination of any defined locally on the annotation and any defined by a dfdl:defineFormat annotation referenced by a local dfdl:ref property.

Fixed Array Element - An array element where XSDL minOccurs is equal to XSDL maxOccurs.

Format annotations - The annotation elements dfdl:format, dfdl:element, dfdl:simpleType, dfdl:group, dfdl:sequence, dfdl:choice, and dfdl:escapeScheme.

> **Comment [m17]:** 2.108 DONE

Format Properties - the attributes on format annotations which specify characteristics of data format.

Framing - framing is the term we use to describe the delimiters, length fields, and other parts of the data stream which are present, and may be necessary to determine the length or position of the content of an element.

Index - The position of an occurrence in a count, starting at 1.

Item - A DFDL information set consists of a number of **information items**; or just *items* for short.

Length - When discussing data items and their representations, the term 'length' is used to refer to the measure of the size of the representation of an item in units of bits, bytes, or characters. The length of an array is the number of bits, bytes, or characters making up its representation, and has nothing to do with the number of occurrences, or dimensionality, of the array. Any element occurrence has length. Only array elements and optional elements have numbers of occurrences other than 1.

Lexical scope - In a DFDL Schema document, the lexical scope of any element is the collection of schema declarations, definitions, and annotations contained within the element textually.

Local properties – Local properties are the properties defined on an annotation in either short, attribute or element form

Logical layer - A DFDL Schema with all the DFDL annotations ignored is an ordinary XSDL schema. The logical structure described by this XSDL is called the DFDL *logical layer*.

Occurrence - An instance of an element in the data, or an item in the DFDL Infoset.

Optional Element - An element declaration or reference where XSDL minOccurs is equal to zero.

Optional Occurrence - An occurrence with an index greater than XSDL minOccurs.

Packed decimal – A physical representation of a decimal and integer number where each digit is packed into one nibble of a byte. There are several variants, some also include a sign nibble and

some include a padding nibble. The term covers all the following enums of the dfdl:binaryNumberRep and dfdl:binaryCalendarRrp properties – 'packed' (IBM 390 packed), 'bcd' (standard binary coded decimals or BCDs) and 'ibm4690Packed' (IBM 4690 packed).

> **Comment [SMH18]:** 2.99. DONE

Potentially represented - an element declaration in the schema describes a *potentially represented* item if that element declaration does not have an inputValueCalc property. Whether the element declaration describes an occurrence that is actually represented or not depends on whether the element declaration is for an optional element or variable array element, and whether the element has a corresponding value in the augmented infoset.

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.

Point of uncertainty - A point of uncertainty occurs in the data stream when there is more than one schema component that might occur at that point.

Representation Property – The properties on component format annotations that affect the representation of the data described by that schema component. These are all the properties with the exception of dfdl:ref and dfdl:hiddenGroupRef.

> **Comment [m19]:** (OPEN) is this the whole list?
>
> See Errata 13 2.126 for change
>
> I'm not sure what the point of enumerating these is here. E.g., dfdl:ref can be put into default scope, but dfdl:hiddenGroupRef cannot, nor can dfdl:inputValueCalc, but we don't say that is a non-property because of it.
>
> *SMH: There is an open WG action on this. (What you say about ref is wrong though – it always acts locally).*

Required Element. An element declaration or reference where XSDL minOccurs is greater than zero.

Required Occurrence - An occurrence with an index less than or equal to XSDL minOccurs.

Required Property – A DFDL property that must have a value. The required properties for each xs:schema component are listed in the Property Precedence tables in section 23.

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.

Scan – Examine the input data looking for delimiters such as separators and terminators.

Schema - The set of all declarations and definitions in the schema, including all included and imported schemas taken together. This includes both the XSDL declarations and definitions, and the DFDL definitions provided in the top-level DFDL annotations.

Schema Definition Order – the order that the schema components are defined in a schema document.

Specified length - An item has specified length when dfdl:lengthKind="implicit", "explicit", or "prefixed".

Speculative Parsing – When the parser reaches a point of uncertainty it attempts to parse each option in turn until one is known to exist or known not to exist.

Statement annotations - The annotation elements dfdl:assert, dfdl:discriminator, dfdl:setVariable, and dfdl:newVariableInstance. Also called DFDL Statements.

Target length - When unparsing, the length (in dfdl:lengthUnits) of an item's representation is the target length. The length of the content corresponding to a logical data value in the infoset may be shorter or longer than the target length, in which case padding or truncation may be necessary to make the logical data content conform to the target length. Rules for when padding and truncation occur, and how they are applied are specific to simple data types, and are controlled by a number of DFDL format properties.

Unpadded length - This is the length of the content of an item of the infoset, prior to any filling or padding which might be introduced due to dfdl:lengthKind="prefixed" or dfdl:lengthKind="explicit". It is equal to or smaller than the target length.

Variable Array Element. An array element where XSDL minOccurs is not equal to XSDL maxOccurs.

Comment [SMH20]: The footnote to this needs reworking due to errata 2.36, as occursCount can now only be an expression.

Comment [m21]: Removed offending sentence and footnote as they were totally unclear to me, and I wasn't sure what point they were trying to make.

4.   The DFDL Information Set (Infoset)

This section defines an abstract data set called the **DFDL Information Set** (**Infoset**). Its purpose is to define the abstract data structure that must be provided:

- To an invoking application by a DFDL parser when parsing DFDL-described data using a DFDL Schema;

- To a DFDL unparser by an invoking application when generating DFDL-described data using a DFDL Schema

The DFDL Infoset contains enough information so that a DFDL schema can be defined that will unparse the infoset and reparse the resultant datastream to produce the same infoset..

There is no requirement for DFDL-described data to be valid in order to have a DFDL information set.

DFDL information sets may be created by methods (not described in this specification) other than parsing DFDL-described data.

A DFDL information set consists of a number of **information items**; or just *items* for short. The information set for any well-formed DFDL-described data will contain at least a document information item and one element information item. An information item is an abstract description of a part of some DFDL-described data: each information item has a set of associated named **members**. In this specification, the member names are shown in square brackets, **[thus]**. The types of information item are listed in Section 4.1 Information Items.

The DFDL Information Set does not require or favor a specific interface or class of interfaces. This specification presents the information set as a modified tree for the sake of clarity and simplicity, but there is no requirement that the DFDL Information Set be made available through a tree structure; other types of interfaces, including (but not limited to) event-based and query-based interfaces, are also capable of providing information conforming to the DFDL Information Set.

The terms "information set" and "information item" are similar in meaning to the generic terms "tree" and "node", as they are used in computing. However, the former terms are used in this specification to reduce possible confusion with other specific data models.

The DFDL Information Set is similar in purpose to the XML Information Set [XMLInfo], however, it is not identical, nor a perfect subset, as there are important differences.

4.1   Information Items

An information set contains two different types of information items, as explained in the following sections. Every information item has members. For ease of reference, each member is given a name, indicated **[thus]**.

4.1.1   Document Information Item

There is exactly one **document information item** in the information set, and all other information items are accessible through the [root] member of the document information item.

There is no specific DFDL schema component that corresponds to this item. It is a concrete artifact describing the information set.

The document information item has the following members:

> **Comment [SMH22]:** Errata 3.7 adds a new member for Unicode BOM. See section 11.
>
> DONE

1. **[root]** The element information item corresponding to the root element declaration of the DFDL Schema.

2. **[dfdlVersion]** String. The version of the DFDL specification to which this information set conforms. For DFDL V1.0 this is 'dfdl-1.0'

3. **[schema]** String. This member is reserved for future use.

4. **[unicodeByteOrderMark]** Enum**.** 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 Byte-order-mark (BOM), and what the value of the mark was. If there is a BOM at the start of the data stream, 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. When unparsing, if this member is not empty and the encoding is UTF-8, UTF-16, or UTf-32, then this member's value is used to determine the specific byte-order mark written, and for UTF-16 and UTF-32, the byte order used when characters are encoded to the output data stream.

4.1.2   Element Information Items

There is an **element information item** for each value parsed from the non-hidden DFDL-described data. This corresponds to an occurrence of a non-hidden element declaration of simple type in the DFDL Schema and is known as a **simple element information item**.

There is an **element information item** for each explicitly declared structure in the DFDL-described data. This corresponds to an occurrence of an element declaration of complex type in the DFDL Schema and is known as a **complex element information item**.

In this information set, as in an XML document, an array is just a set of adjacent elements with the same name and namespace. (To represent the array explicitly, introduce a new complex type element to contain the array elements only.)

One of the element information items is the [root] member of the document information item, corresponding to the root element declaration of a DFDL Schema, and all other element information items are accessible by recursively following its [children] member.

An element information item has the following members:

1. **[namespace]** String. The namespace, if any, of the element. If the element does not belong to a namespace, the value is the empty string.

2. **[name]** String. The local part of the element name.

3. **[document]** The document information item representing the DFDL information set that contains this element. This element is empty except in the root element of an information set.

4. **[datatype]** String. The name of the XML Schema 1.0 built-in simple type to which the value corresponds. DFDL supports a subset of these types listed in section 5.1 DFDL Subset of XML Schema. In a complex element information item this member has no value.

5. **[dataValue]** The value in the value space (as defined by XML Schema Part 2: Datatypes [XSDLV1]) of the [datatype] member or special value *nil***.** In a complex element information item this member has no value.

   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.

6. **[children]** An ordered set of zero or more element information items. The order they appear in the set is the order implied by the DFDL Schema. 'Ordered set' is not formally defined here, but two operations are assumed: 'count' gives the number of information items, and 'at (index)' gives the element at ordinal position 'index' starting from 1. In a simple element information item this member has no value. In a document information item this member contains exactly one element information item.

7. **[parent]** The complex element information item which contains this information item in its [children] member. In the root element of an information set this member is empty.

8. **[schema]** String. A reference to a schema component associated with this information item, if any. If not empty, the value must be an absolute or relative Schema Component Designator (SCD).

9. **[valid]** Boolean[4]. True if the element is valid as determined by a DFDL implementation that performs validation checking. A complex element information item is not valid if any of its [children] are not valid. Empty if validation is not enabled.

10. **[unionMemberSchema]**[5] String. For simple element information items, this member contains 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 the **[valid]** or **[unionMemberSchema]** members are ignored. However, in the augmented infoset which is built during the unparse operation **[valid]** will have a value, and **[unionMemberSchema]** may have a value.

4.2   "No Value"

Some members may sometimes have the value *no value*, and it is said that such a member has no value. This value is distinct from all other values. In particular it is distinct from the empty string, the empty set, and the empty list, each of which simply has no members, and it is also distinct from the special value *nil*.

4.3   DFDL Information Item Order

On parsing and unparsing information items will be presented in the order they are defined in the DFDL Schema.

4.4   DFDL Infoset Object model

By way of illustration, the DFDL information set is presented below as an object model using a Unified Modeling Language (UML) class diagram, augmented using the Object Constraint Language (OCL) [http://www.omg.org/technology/documents/modeling_spec_catalog.htm].

The structure of the information set follows the Composite design pattern. In case of inconsistency or ambiguity, the preceding discussion takes precedence.

---

[4] The purpose of this member is to support construction of a W3C standard Post Schema Validation Infoset (PSVI) from a DFDL Infoset.

[5] Also to support PSVI construction.

Comment [SMH26]: Errata 2.67 changes this

DONE

Comment [SMH27]: Errata 3.13 changes this and adds control over behaviour.

DONE

Comment [m28]: I believe this language is fixed w.r.t. 3.13. This section is about the infoset and doesn't need to discuss encoding errors.

Comment [SMH29]: Should be 'complex'

DONE

Comment [SMH30]: This is a Boolean so how can it be empty. Should this be 'no value' ?

Comment [SMH31]: See comment above.

Comment [SMH32]: [valid] will have a value. [unionMemberSchema] can have a value.

DONE

DFDL is able to describe the format of the physical representation for data whose structure conforms to this model. Note that this model allows hierarchically nested data, but does not allow representation of arbitrary connected graphs of data objects.
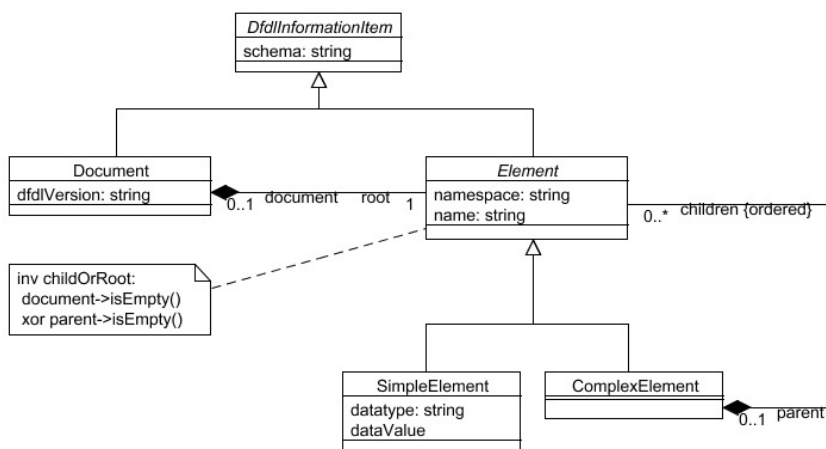
Figure 1 DFDL Infoset Object Model

4.5   DFDL Augmented Infoset

When unparsing, one begins with the DFDL schema and conceptually with the logical infoset. As the values of items are filled in by defaulting, and by use of the dfdl:outputValueCalc property (including on hidden items) (see section 117 Calculated Value Properties.), these new item values augment the infoset. The resulting infoset is called the augmented infoset.

An element declaration in the schema describes a *potentially represented* item if that element declaration does not have a dfdl:inputValueCalc property (see section 17 Calculated Value Properties.). Whether the element declaration describes an item that is actually represented or not depends on whether the element declaration is for an optional element or variable array element, and whether the element has a corresponding value in the augmented infoset.

In expressions, the function dfdl:representationLength() can be called to determine the length of the representation of an item including all content and framing. If an element declaration is not potentially represented, then dfdl:representationLength() is defined to return 0.

When unparsing, an element declaration and the infoset are considered as follows below. An implementation may use any technique consistent with this algorithm:

a)   If the element declaration has a dfdl:outputValueCalc property then the expression which is the dfdl:outputValueCalc property value is evaluated and the resulting value becomes the value of the element item in the augmented infoset. Any pre-existing value for the infoset item is superseded by this new value.

References to other augmented infoset items from within the outputValueCalc expression must obtain their values from the augmented infoset directly (when the value is already present) or by recursively using these methods (a) and (b) as needed.

b)   If the element declaration has no corresponding value in the augmented infoset, and the element declaration is for a *required* occurrence, and it *has a default value specified*, then an element item having the default value is created in the augmented infoset.

c)   If any Infoset item's value is requested recursively as a part of (a) above and (a) does not apply, and the corresponding value is not present, and (b) does not apply then it is a processing error.

Given this augmented infoset, then if the potentially represented element declaration has a corresponding infoset item then that item is converted to its representation according to its DFDL properties. If the element declaration is for a required occurrence, and there is no value in the augmented infoset then it is a processing error.

Because rule (a) above is used even if the augmented infoset item already exists and has a value, it is possible for an outputValueCalc expression to be evaluated multiple times. DFDL implementations are free to cache values and avoid this repeated evaluation for efficiency, as the semantics of DFDL require that the outputValueCalc expression return the same value every time it is evaluated.

5. DFDL Schema Component Model

When using DFDL, the format of data is described by means of a *DFDL Schema*.

The DFDL Schema Component Model is shown in conceptual UML in Figure 2. First we show the model for elements, groups and the top of the type hierarchy.

The shaded boxes have direct corresponding element syntax and therefore appear in DFDL schema
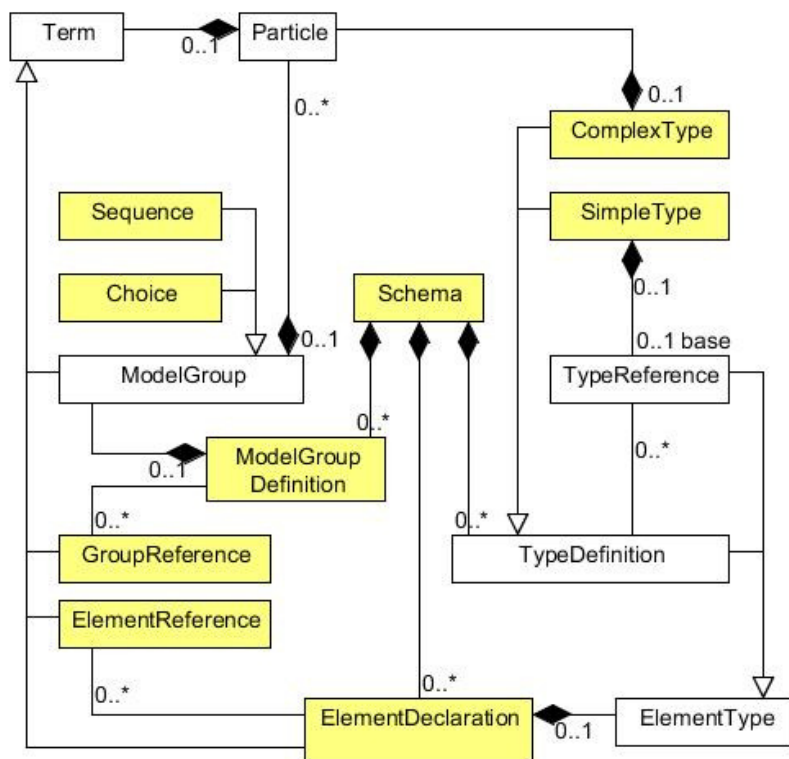
Figure 2 DFDL Schema UML diagram

The simple types are shown in Figure 3. The graph shows all the types defined by XML Schema version 1.0, and the subset of these types supported by DFDL are shown as shaded.

DFDL built-in types
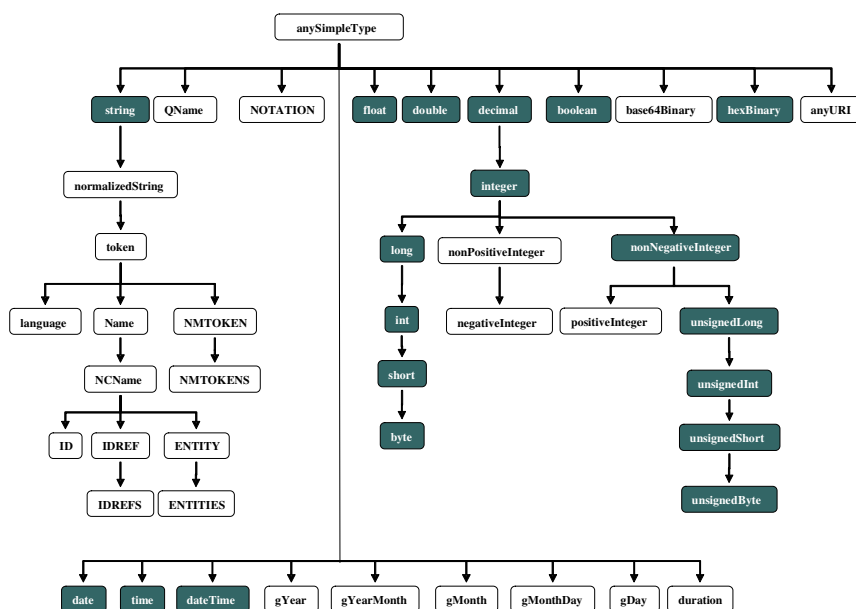
Figure 3 DFDL simple types

These types are defined as they are in XML Schema, with exceptions for:

> String – In DFDL a string can contain any character codes. None are reserved. (Including the character with character code U+0000, which is not permitted in XML documents.)

Each object defined by a class in the above UML is called a *DFDL Schema component*.

We express the DFDL Schema Model using a subset of the XML Schema Description Language (XSDL). XSDL provides a standardized schema language suitable for expressing the DFDL Schema Model.

A DFDL Schema is an XML schema containing only a restricted subset of the constructs available in full W3C XML Schema Description Language. Within this XML schema, special DFDL annotations are distributed that carry the information about the data's format or representation.

A DFDL Schema is a valid XML schema. However, the converse is not true since the DFDL Schema Model does not include many concepts that appear in XML schema.

5.1   DFDL Subset of XML Schema

The DFDL subset of XSDL is a general model for hierarchically-nested data. It avoids the XSDL features used to describe the peculiarities of XML as a syntactic textual representation of data, and features that are simply not needed by DFDL.

The following lists detail the similarities and differences between general XSDL and this subset.

DFDL Schemas consist of:

- Standard XSDL namespace management

- Standard XSDL import and management for multiple file schemas

- Local element declarations with dimensionality via maxOccurs and minOccurs.

- Global element declarations

- ComplexType definitions with empty or element-only content models.

- DFDL appinfo annotations describing the data format

- These simple types: string, float, double, decimal, integer, long, int, short, byte, nonNegativeInteger, unsignedLong, unsignedInt, unsignedShort, unsignedByte, boolean, date, time, dateTime, hexBinary

- These facets: minLength, maxLength, minInclusive, maxInclusive, minExclusive, maxExclusive, totalDigits, fractionDigits, enumeration, pattern (for xs:string type only)

- Fixed values

- Default values

- 'sequence' model groups (without minOccurs and maxOccurs)

- 'choice' model groups (without minOccurs and maxOccurs)

- Simple type derivations derived by restriction from the allowed built-in types

- Reusable Groups: named model group definitions can only contain one model group

- Element references with dimensionality via maxOccurs and minOccurs.

- Group references without dimensionality

- Nillable="true"

- Appinfo annotations for sources other than DFDL are permitted and ignored

- Unions; the memberTypes must be derived from the same simple type. DFDL annotations are not permitted on union members.[6]

- XML Entities

> **Comment [SMH33]:** Errata 3.10 allows nillable complex types
>
> DONE

---

[6] The purpose of unions is to allow multiple constraints via facets such as multiple independent range restrictions on numbers. This enhances the ability to do rich validation of data.

Note: xs:nonNegativeInteger is treated as an unsigned xs:integer.

The following constructs from XML Schema are not used as part of the DFDL Schema Model of DFDL v1.0 schemas; however, they are all reserved[7] for future use since the data model may be extended to use them in future versions of DFDL:

- Attribute declarations (local or global)

- Attribute references

- Attribute group definitions

- Complex type derivations where the base type is not xs:anyType.

- Complex types having mixed content models or simple content models

- List simple types

- Union simple types where the member types are not derived from the same simple type.

- These atomic simple types: normalizedString, token, Name, NCName, QName, language, positiveInteger, nonPositiveInteger, negativeInteger, gYear, gYearMonth, gMonth, gMonthDay, gDay, ID, IDREF, IDREFS, ENTITIES, ENTITY, NMTOKEN, NMTOKENS, NOTATION, anyURI, base64Binary

- maxOccurs and minOccurs on model groups

- minOccurs = 0 on branches of xs:choice model groups

- Identity Constraints

- Substitution Groups

- xs:all groups

- xs:any element wildcards

- Redefine - This version of DFDL does not support xs:redefine. DFDL schemas must not contain xs:redefine directly or indirectly in schemas they import or include.

- whitespace facet

- Recursively-defined types and elements (defined by way of type, group, or element references)

5.2   XSD Facets, min/maxOccurs, default, and fixed

XSD element declarations and references can carry several attributes and properties that express constraints on the described data. These constraints are mainly used for validation. These attributes and properties include:

- the facets

- minOccurs, maxOccurs

- default

- fixed

The facets and the types they are applicable to are:

---

[7] By reserved we mean that conforming DFDL v1.0 implementations MAY NOT assign semantics to them.

> **Comment [SMH34]:** Errata 3.10 allows this DONE

> **Comment [m35]:** Errata 2.111 Done

- minLength maxLength (for types xs:string, and xs:hexBinary)
- pattern (all types)
- enumeration (all types except xs:boolean)
- maxInclusive, maxExclusive, minExclusive, minInclusive (for types xs:float, xs:double, xs:date, xs:time, xs:dateTime, xs:decimal and all integer types descending from xs:decimal in Figure 3)
- totalDigits (for type xs:decimal and all integer types descending from xs:decimal in Figure 3)
- fractionDigits (for type xs:decimal)

The facets (but not maxOccurs nor minOccurs) are also checked by the dfdl:checkConstraints DFDL expression language function.

The following sections describe these in more detail.

5.2.1   MinOccurs and MaxOccurs

The XSDL minOccurs property is used:

- To determine if an element declaration or reference is an array, an optional element, or neither.
- For some values of the property dfdl:occursCountKind, to determine the required minimum number of occurrences of an array both when parsing and unparsing.
- If validating, to determine the minimum valid number of occurrences of an array both when parsing and unparsing.

The XSDL maxOccurs property is used:

- To determine if an element declaration or reference is an array, an optional element, or neither.
- When dfdl:occursCountKind="fixed", then the XSDL maxOccurs value is the fixed number of occurrences of the array element, which is then called a Fixed Array Element. It is a schema definition error if XSDL minOccurs is not equal to XSDL maxOccurs.
- When dfdl:occursCountKind="implicit" then XSDL maxOccurs value (if not unbounded) is used to determine the maximum number of element occurrences both when parsing or unparsing.
- If validating, to determine the maximum valid number of occurrences of an array both when parsing and unparsing.

For some values of dfdl:occursCountKind such as 'implicit', it is a processing error when an array is found to have a number of occurrences not conforming to XSDL minOccurs in the absence of a default value specification. For other values of dfdl:occursCountKind such as 'parsed', it is only a validation error if an array is found to have fewer than XSDL minOccurs occurrences. See Section 16, Properties for Array Elements and Optional Elements, for more details.

5.2.2   MinLength, MaxLength

These facets are used:

- When dfdl:lengthKind="implicit" and type is xs:string or xs:hexBinary. In that case the length is given by the value of xs:maxLength. In this case xs:minLength is required to be equal to maxLength (schema definition error otherwise).
- For validation of variable length string elements.

Comment [SMH36]: Revisit this section once errata exists for action 140

I revisited this and added the detail about OCK implicit.

I *think* that's all.

DONE

5.2.3   MaxInclusive, MaxExclusive, MinExclusive, MinInclusive, TotalDigits, FractionDigits

- Used for validation only

The format of numbers is not derived from these facets. Rather dfdl properties are used to specify the format.

5.2.4   Pattern

- Allowed only on elements of type xs:string or derived from it.

- Used for validation only

It is important to avoid confusion of the pattern facet with other uses of regular expressions that are needed in DFDL. For example, to determine the length of an element by regular expression matching.

Note: in XSD, pattern is about the lexical representation of the data, and since all is text there, everything has a lexical representation. In DFDL only strings are guaranteed to have a lexical and logical value that is identical.

5.2.5   Enumeration

Enumerations are used to provide a list of valid values in XSD.

- Used for validation only

Note: in DFDL we do not use XSD enumeration as a means to define symbolic constants. These are captured using dfdl:defineVariable constructs so they can be referenced from expressions.

5.2.6   Default

The 'default' attribute is used:

- To provide the logical value of a required element while parsing when the element is missing. See 13.15 Nil Value Processing

- To provide the logical value of a required element when unparsing when element is missing. See 13.15 Nil Value Processing

5.2.7   Fixed

The 'fixed' attribute is used:

- To constrain the logical value of an element when validating.

- To provide the logical value of a required element while parsing when the element is missing. See 13.15 Nil Value Processing

- To provide the logical value of a required element when unparsing when element is missing. See 13.15 Nil Value Processing

6. DFDL Syntax Basics

Using DFDL, a data format is described by placing special annotations at various positions within an XML schema. This XML schema conveys the basic structure of the data format, while the annotations fill in the detail. Annotations are used to describe aspects such as the file encoding and byte ordering, as well as declaring variables for reference elsewhere, and specifying properties that govern the capabilities of the DFDL processor. A DFDL processor requires these annotations, along with the structural information of the enclosing XML schema, to make sense of the physical data model.

6.1   Namespaces

The xs:appinfo source URI http://www.ogf.org/dfdl/ is used to distinguish DFDL annotations from other annotations.

The element and attribute names in the DFDL syntax are in a namespace defined by the URI http://www.ogf.org/dfdl/dfdl-1.0/. All symbols in this namespace are reserved. DFDL implementations may not provide extensions to the DFDL standard using names in this namespace. Within this specification, the namespace prefix for DFDL is "dfdl" referring to the namespace http://www.ogf.org/dfdl/dfdl-1.0/.

Attributes on DFDL annotations that are not in the DFDL namespace or or in no namespace are ignored.

A DFDL Schema document contains XML schema annotation elements that define and assign names to parts of the format specification. These names are defined using the target namespace of the schema document where they reside, and are referenced using QNames in the usual manner. A DFDL schema document can include or import another schema document, and namespaces work in the usual manner for XML schema documents. The *schema* is the schema including all additional schemas referenced through import and include. Generally, in this specification, when we refer to the DFDL Schema we mean the schema. When we refer to a specific document we will use the term DFDL Schema document.

> **Comment [SMH38]:** 2.12. Clarified. DONE.

6.2   The DFDL Annotation Elements

> **Comment [m39]: 2.109**. *Done*

DFDL annotations must be positioned specifically where DFDL annotations are allowed within an XML schema document. These positions are known as *annotation points*. When an annotation is positioned at an annotation point, it binds some additional information to the schema component that encloses it. The description of a data format is achieved by correctly placing annotations on the structural components of the schema.

DFDL specifies a collection of annotations for different purposes. They are organized into three different annotation types: Format Annotations, Statement Annotations, and Defining Annotations

> **Comment [SMH40]:** 2.108. DONE

At any single annotation point of the schema there can be only one format annotation, but there can be several statement annotations although there are rules about which of those are allowed to co-exist as well which will be described in sections about those specific annotation types.

Annotation Type	Annotation Element(s)	Description
Format Annotation	choice	Defines the physical data format properties of an xs:choice group. See section 7.1.1
	element	Defines the physical data format properties of an xs:element and xs:element reference. See section 7.1.1

> **Comment [m41]:** (OPEN)
>
> Throughout the document, make all the table headers consistent in font, size, face, justificaiton, and background color.

	format	Defines the physical data format properties for multiple DFDL schema constructs. Used on an xs:schema and as a child of a dfdl:defineFormat annotation. This includes aspects such as the encodings, separators, and many more. See section 7.1
	group	Defines the physical data format properties of an xs:group reference. See section 7.1.1
	property	Used in the syntax of format annotations. See section 7.1.3.2.
	sequence	Defines the physical data format properties of an xs:sequence group. See section 7.1.1
	simpleType	Defines the physical data format properties of an xs:simpleType. See section 7.1.1
	escapeScheme	Defines the scheme by which quotation marks and escape characters can be specified. This is for use with delimited text formats. See section 7.6
Statement Annotation	assert	Defines a test to be used to ensure the data are well formed. Assert is used only when parsing data. See section 7.3
	discriminator	Defines a test to be used when resolving a point of uncertainty such as choice branches, optional elements, or variable array elements. A dfdl:discriminator is used only when parsing data to resolve the point of uncertainty to one of the alternatives. See section 7.4
	newVariableInstance	Creates a new instance of a variable. See section 7.8
	setVariable	Sets the value of a variable whose declaration is in scope See section 7.9
Defining Annotation	defineEscapeScheme	Defines a named, reusable escapeScheme See section 7.5
	defineFormat	Defines a reusable data format by collecting together other annotations and associating them with a name that can be referenced from elsewhere. See section 7.2
	defineVariable	Defines a variable that can be referenced elsewhere. This can be used to communicate a parameter from one part of processing to another part. See section 7.7

Table 1 - DFDL Annotation Elements

> **Comment [m42]:** This table Modified, but is not marked with change bars. (Many table edits cannot create change bars). First column added to group the annotations into the 3 annotation types.

6.3   DFDL Properties

Properties on DFDL annotations may be one or more of the following types

- DFDL string literal
  The property value is a string that represents a sequence of literal bytes or characters which appear in the data stream.

Comment [SMH43]: Suggest removing as it needs the full explanation as given in 6.3.1 to be unambiguous.

DONE

- DFDL expression
  The property value is a DFDL subset XPath 2.0 expression that returns a value derived from other property values and/or from the DFDL infoset.

- DFDL regular expression
  The property value is a regular expression that can be used as a pattern to calculate the length of an element by applying that pattern to the sequence of literal bytes or characters which appear in the data stream.

- Enumeration
  The property value is one of the allowed values listed in the property description. An enumeration is of type string unless otherwise stated.

- Logical Value.
  The property value is a string that describes a logical value. The type of the logical value is one of the XML schema simple types. The string must conform to the XML schema lexical representation for the type.

Comment [SMH44]: Errata 2.10. DONE

- QName
  The property value is an XML Qualified Name as specified in "Namespaces in XML" [XMLNS10]

Some properties accept a list or union of types

- List of DFDL String Literals or Logical Values
  The property value is a space-separated list of the specified type. When parsing, if more than one string literal in the list matches the portion of the data stream being evaluated then the longest matching value in the list must be used. When unparsing, the first value in the list must be used. String literals containing whitespace or string literals representing the empty string must use character class entities in their syntax.

Comment [m45]: Added.

- Union of types and expressions.
  The property value is a union of DFDL expression and exactly one of the other types. The expression must resolve to a value of the other type.

- Union of types.
  The property value is a union of two or more types. The type is dependent on the value of another property. For example dfdl:nilValue can be a List of DFDL String Literals or a List of Logical Values depending on dfdl:nilKind

6.3.1   DFDL String Literals

Comment [SMH46]: Errata 2.11 clarifies the use of white space.
Errata 2.64 clarifies the use of empty string.

DONE

And

DONE

DFDL String Literals represent a sequence of literal bytes or characters which appear in the data stream. This presents the following challenges

- the literal characters in the data stream might not be in the same encoding as the DFDL schema

- it may be necessary to specify a literal character which is not valid in an XML document

- it may be necessary to specify one or more raw byte values

A DFDL string literal can describe any of the following types of literal data in any combination:

- a single literal character in any encoding

- a string of literal characters in any encoding

- a bi-directional character string

- one or more characters from a set of related characters (e.g. end-of-line characters)

- a literal byte value

A DFDL string literal is therefore able to describe any arbitrary sequence of bytes and characters.

Empty Strings: Empty string is not allowed as a DFDL string literal value unless explicitly stated otherwise in the description of a property. In this case the use of empty string provides some property specific behavior different from simply using the empty string as a value. When the empty string is to be used as a value, the entity %ES; must be used in the corresponding DFDL string literal.

Whitespace: When whitespace must be used as part of a property value, the DFDL string literal must use entities (such as %WSP;) to represent the whitespace. (This allows a property to represent lists of DFDL string literals by using literal spaces to separate list elements.)

6.3.1.1   Character strings in DFDL String Literals

A literal string in a DFDL Schema is written in the character set encoding specified by the XML directive that begins all XML documents:

```
<?xml version="1.0" encoding="UTF-8" ?>
```

In this example, the DFDL schema is written in UTF-8, so any literal strings contained in it, and particularly string literals found in its representation property bindings in the format annotations, are expressed in UTF-8.

However, these strings are being used to describe features of text data that are commonly in other character set encodings. For example, we may have EBCDIC data that is comma separated. A comma in EBCDIC has a single-byte code unit of 0x6B in the data, the numeric value of which does not correspond to the Unicode character code for comma which is U+002C. However, when we indicate that an item is "," (comma) separated and we specify this using a string literal along with specifying the 'encoding' property to be 'ebcdic-cp-us' then this means that the data are separated by EBCDIC commas regardless of what character set encoding is used to write the DFDL Schema.

```
<?xml version="1.0" encoding="UTF-8">
<xs:schema ... >
    ...
    <dfdl:format encoding="ebcdic-cp-us" separator=","/>
    ...
</xs:schema>
```

When a DFDL processor uses the separator expressed in this manner, the string literal "," is *translated* into the character set encoding of the data it is separating as specified by the encoding representation property. Hence, in this case we would be searching the data for a character with codepoint 0x6B (the EBCDIC comma), not a UTF-8 or Unicode (0x2C) comma which is what exists in the DFDL schema document file.

Character strings can include bidirectional data.

6.3.1.2   DFDL Character Entities in String Literals

DFDL character entities specify a single Unicode character and provide a convenient way to specify code points that appear in the data stream but would be difficult to specify in XML strings. For example, common non-printable characters or code points, such as 0x00, that are not valid in XML documents. DFDL entities are based on XML entities, which can also be used in a DFDL schema.

DfdlCharEntity	::=	DfdlEntity |
		DecimalCodePoint |
		HexadecimalCodePoint
DfdlEntity	::=	'%' DfdlEntityName ';'
DfdlEntityName	::=	'NUL'|'SOH"|'STX'|'ETX'|
		'EOT'|'ENQ'|'ACK'|'BEL'|
		'BS'|'HT'|'LF'|'VT'|'FF'|
		'CR'|'SO'|'SI'|'DLE'|
		'DC1'|'DC2'|'DC3'|'DC4'|
		'NAK'|'SYN'|'ETB'|'CAN'|
		'EM'|'SUB'|'ESC'|'FS'|
		'GS'|'RS'|'US'|'SP'|
		'DEL'|'NBSP'|'NEL'|'LS'
DecimalCodePoint	::=	'%#' [0-9]+ ';'
HexadecimalCodePoint	::=	'%#x' [0-9a-fA-F]+ ';'

Table 2 DFDL Character Entity syntax

Using %% inserts a single literal "%" into the string literal. This "%" is subject to character set encoding translation as is any other character.

A HexadecimalCodePoint provides a hexadecimal representation of the character's code point in ISO/IEC 10646.

A DecimalCodePoint provides a decimal representation of the character's code point in ISO/IEC 10646.

A dfdlEntityName is one of the mnemonics given in the following tables.

Mnemonic	Meaning	Unicode Character Code
NUL	null	U+0000
SOH	start of heading	U+0001
STX	start of text	U+0002
ETX	end of text	U+0003
EOT	end of transmission	U+0004
ENQ	enquiry	U+0005
ACK	acknowledge	U+0006
BEL	bell	U+0007
BS	backspace	U+0008
HT	horizontal tab	U+0009
LF	line feed	U+000A
VT	vertical tab	U+000B

FF	form feed	U+000C
CR	carriage return	U+000D
SO	shift out	U+000E
SI	shift in	U+000F
DLE	data link escape	U+0010
DC1	device control 1	U+0011
DC2	device control 2	U+0012
DC3	device control 3	U+0013
DC4	device control 4	U+0014
NAK	negative acknowledge	U+0015
SYN	synchronous idle	U+0016
ETB	end of transmission block	U+0017
CAN	cancel	U+0018
EM	end of medium	U+0019
SUB	substitute	U+001A
ESC	escape	U+001B
FS	file separator	U+001C
GS	group separator	U+001D
RS	record separator	U+001E
US	unit separator	U+001F
SP	space	U+0020
DEL	delete	U+007F
NBSP	no break space	U+00A0
NEL	Next line	U+0085
LS	Line separator	U+2028

Table 3 DFDL Entities

6.3.1.3   DFDL Character Class Entities in DFDL String Literals

Comment [m47]: **2.94**. *Sections 6.3.1.3, 12.2*. DONE

The following DFDL character classes are provided to specify one or more characters from a set of related characters.

DfdlCharClass	::=	'%' DfdlCharClassName ';'
DfdlCharClassName	::=	'NL' | 'WSP' | 'WSP*' | 'WSP+' | 'ES'

Table 4 DFDL Character Class Entity syntax

Mnemonic	Meaning	Unicode Character Code(s)
NL	Newline On parse any one of the single	U+000A LF

		characters CR, LF, NEL or LS or the character combination CRLF. On unparse the value of the dfdl:outputNewLine property is output, which must specify one of the single characters %CR;, %LF;, %NEL;, or %LS; or the character combination %CR;%LF;.	U+000D CR U+000D U+000A CRLF U+0085 NEL U+2028 LS
	WSP	Single whitespace On parse any white space character On unparse a space (U+0020) is output	U+0009-U+000D (Control characters) U+0020 SPACE U+0085 NEL U+00A0 NBSP U+1680 OGHAM SPACE MARK U+180E MONGOLIAN VOWEL SEPARATOR U+2000-U+200A (different sorts of spaces) U+2028 LSP U+2029 PSP U+202F NARROW NBSP U+205F MEDIUM MATHEMATICAL SPACE U+3000 IDEOGRAPHIC SPACE
	WSP*	Optional Whitespaces On parse whitespace characters are ignored On unparse nothing is output	Same as WSP
	WSP+	Whitespaces On parse one or more whitespace characters are ignored. It is an processing error if no whitespace character is found On unparse a space (U+0020) is output	Same as WSP
	ES	Empty String Used in space separated lists when empty string is one of the values (may only be used for the nilValue property)	

Table 5 **DFDL Character Class Entities**

6.3.1.4   DFDL Byte Value Entities in DFDL String Literals

DFDL byte value entities provide a way to specify a single byte as it appears in the data stream without any character set encoding translation. To specify a string of byte values, a sequence of two or more byte value entities must be used.

ByteValue	::=	'%#r' [0-9a-fA-F]{2} ';'

Table 6 DFDL Byte Value Entity syntax

6.3.2   DFDL Expressions

Some DFDL properties allow DFDL expressions [see Section 23 Expression language] to be used so that the property can be set dynamically at processing-time.

The general syntax of expressions is "{" expression "}"

The rules for recognizing DFDL expressions are

- Must start with a '{' in the first position and end with '}' in the last position.

- '{' in any position other than the first is treated as a literal

- '}' in any position other than the last position is treated as a literal.

- '{{' as the first characters are treated as the literal '{' and not as the start of a DFDL expression.

DFDL expressions reference other items in the infoset or augmented infoset using absolute or relative paths. Relative paths are evaluated when the component containing the expression is referenced not when it is declared. For example a global element may have a DFDL property which is an expression that contains a relative path to another element. The relative path is evaluated when the global element is referenced from an element reference.

DFDL expressions that are used to provide the value of DFDL properties in the dfd:format annotation on the top level xs:schema declaration MAY NOT contain relative paths.

6.3.3   DFDL Regular Expressions

The DFDL lengthPattern property expects a regular expression to be specified. The DFDL Regular Expression language is defined in the section 24 DFDL Regular Expressions.

6.3.4   Enumerations in DFDL

Some DFDL properties accept an enumerated list of valid values. It is a schema definition error if a value other than one of the enumerated values is specified. The case of the specified value must match the enumeration. An enumeration is of type string unless otherwise stated.

> **Comment [SMH48]:** Errata 2.49 clarifis this escaping rule. **DONE**

> **Comment [SMH49]:** Currently being discussed under action 168. Errata will be taken to clarify wording.
>
> Action 168 closed. DONE

7. Syntax of DFDL Annotation Elements

This section describes the syntax of each of the DFDL annotation elements along with discussion of their basic meanings.

The DFDL annotation elements are listed in **Table 1 - DFDL Annotation Elements**

7.1   Component Format Annotations

A data format can be 'used' or put into effect for a part of the schema by use of the component format annotation elements.

There are specific annotations for each type of schema component that supports only the representation properties applicable to that component. The table below gives the specific annotation for each schema component.

Schema component	DFDL annotation
xs:choice	dfdl:choice
xs:element	dfdl:element
xs:element reference	dfdl:element
xs:group reference	dfdl:group
xs:schema	dfdl:format
xs:sequence	dfdl:sequence
xs:simpleType	dfdl:simpleType

Table 7 DFDL Component Format Annotations

In addition the dfdl:format annotation is used inside a dfdl:defineFormat annotation to define a named reusable set of representation properties that can be referenced from any component specific format annotation or from other named format definitions.

A dfdl:format annotation at the top level of a schema, that is as an annotation child element on the xs:schema, provides a set of default properties for the lexically enclosed schema document. See 8.1 Providing Defaults for DFDL properties.

Example of DFDL component format annotation:

```
<xs:schema ...>
  ...
  <xs:element name="root">
    <xs:annotation>
      <xs:appinfo source="http://www.ogf.org/dfdl/">
        <dfdl:element ref="aBaseConfig"
                      representation="text"
                      encoding="UTF-8"/>
      </xs:appinfo>
    </xs:annotation>
  </xs:element>
  ...
</xs:schema>
```

7.1.1   Syntax of Component Format Annotations

Property Name	Description
ref	QName
	Reference to a named dfdl:defineFormat annotation that provides a reusable set of DFDL format properties
	The ref property is always local to the component format annotation on which it is used, even when specified on a format annotation on the xs:schema element.
	See 7.2 dfdl:defineFormat - Reusable Data Format Definitions
	Annotation: dfdl:element, dfdl:simpleType, dfdl:sequence, dfdl:choice, dfdl:group
representation properties	All other attributes on format annotation elements are representation property bindings. These are defined in sections starting with section 9 DFDL Processing Introduction
	Annotation: dfdl:element, dfdl:simpleType, dfdl:sequence, dfdl:choice, dfdl:group

Table 8 Component Annotation Syntax

7.1.2   Ref Property

A named, reusable, dfdl:defineFormat definition is used by referring to its name from a format annotation using the 'ref' attribute. For example:

```
<dfdl:element ref="reusableDef" encoding="ebcdic-cp-us" />
```

The behavior of this dfdl:defineFormat definition is as if all representation properties defined by the named dfdl:defineFormat definition were instead written directly on this format annotation; however, these are superseded by any representation properties that are defined here such as the encoding property in the example above.

7.1.3   Property Binding Syntax

The format properties may be specified in one of three forms:

1. Attribute form
2. Element form
3. Short form

A DFDL property may be specified using any form with the following exceptions

- The ref property may be specified in attribute or short form
- The escapeSchemeRef property may be specified in attribute or short form
- The hiddenGroupRef property may be specified in attribute or short form
- The prefixLengthType property may be specified in attribute or short form
- Short form is not allowed on the xs:schema element..

It is a schema definition error if the same property is specified in more than one form in the resolved set of annotations for an annotation point.

7.1.3.1   Property Binding Syntax: Attribute Form

Within the format annotation elements are bindings for properties of the form:

Property=`'`*Value*`'`

For example:

```
  <xs:annotation>
    <xs:appinfo source="http://www.ogf.org/dfdl/">
      <dfdl:format encoding="utf-8" separator="%NL;"/>
  </xs:appinfo>
 </xs:annotation>
```

The *Property* is the name of the property. The *Value* is an XML string literal corresponding to a value of the appropriate type.

7.1.3.2   Property Binding Syntax: Element Form

The representation properties can sometimes have complex syntax, so an element form for representation property bindings is provided as element content within the format element content model. This is provided to ease syntactic expression difficulties. The element is called dfdl:property and it has one attribute 'name' which provides the name of the property.

For example:

```
  <xs:annotation>
    <xs:appinfo source="http://www.ogf.org/dfdl/">
      <dfdl:format>
        <dfdl:property name='encoding'>utf-8</dfdl:property>
        <dfdl:property name='separator'>%NL;</dfdl:property>
      </dfdl:format>
    </xs:appinfo>
 </xs:annotation>
```

Element form is mostly used for properties that themselves contain the quotation mark characters and escape characters so that they can be expressed without concerns about confusion with the XSDL syntax use of these same characters. CDATA encapsulation can be used so as to allow malformed XML and mismatched quotes to be easily used as representation property values:

```
    <dfdl:property name='initiator'><[CDATA[<!-- ]]></dfdl:property>
```

7.1.3.3   Property Binding Syntax:Short Form

To save textual clutter, short-form syntax for format annotations is also allowed on xs:element, xs:sequence, xs:choice, xs:group (for group references only), and xs:simpleType schema elements. (The xs:schema element cannot carry short-form annotations). Attributes which are in the namespace 'http://www.ogf.org/dfdl/dfdl-1.0/' and whose local name matches one of the DFDL representation properties are assumed to be equivalent to specific DFDL attribute form annotations.

> **Comment [SMH50]:** Errata 2.59 disallows short form on xs:schema.
>
> DONE

For example the two forms below are equivalent in that they describe the same data format. The first is the short form of the second:

```
<xs:element name="elem1">
  <xs:complexType>
    <xs:sequence dfdl:separator="%HT;" >
       ...
    </xs:sequence>
  </xs:complexType>
</xs:element>

<xs:element name="elem2">
```

```
  <xs:complexType>
    <xs:sequence>
      <xs:annotation><xs:appinfo source="http://www.ogf.org/dfdl/">
        <dfdl:sequence separator="%HT;" />
      </xs:appinfo></xs:annotation>
      ...
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

Another example:

```
<xs:sequence dfdl:separator=",">
  <xs:element name="elem1" type="xs:int" maxOccurs="unbounded"
                        dfdl:representation="text"
                        dfdl:textNumberRep="standard"
                        dfdl:initiator="["
                        dfdl:terminator="]"/>

  <xs:element name="elem2" type="xs:int" maxOccurs="unbounded">
    <xs:annotation><xs:appinfo source="http://www.ogf.org/dfdl/">
      <dfdl:element representation="text"
                    textNumberRep="standard"
                    initiator="["
                    terminator="]"/>
    </xs:appinfo></xs:annotation>
  </xs:element>
</xs:sequence>
```

Because short form syntax is not allowed on the xs:schema element, an attribute form dfdl:format annotation must be used instead.

Comment [m51]: Errata 2.59 done

7.1.4   Empty String as a Representation Property Value

DFDL provides no mechanism to un-set a property. Setting a representation property's value to the empty string doesn't remove the value for that property, but sets it to the empty string value. This may not be appropriate as a value for certain properties.

For example, in delimited text representations, it is sensible for the separator to be defined to be the empty string. This turns off use of separator delimiters. For many other string-valued properties, it is a schema definition error to assign them the empty string value. For example, the character set encoding property (dfdl:encoding) cannot be set to the empty string.

7.2   dfdl:defineFormat - Reusable Data Format Definitions

One or more dfdl:defineFormat annotation elements can appear within the annotation children of the xs:schema element. DFDL defining annotation elements may only appear as annotation children of the xs:schema element.

The order of their appearance does not matter, nor does their position relative to other non-annotation children of the xs:schema.

Each dfdl:defineFormat has a required name attribute.

The construct creates a named data format definition. The value of the name attribute is of XML type NCName. The format name will become a member of the schema's target namespace. These names must be unique within the namespace.

If multiple format definitions have the same 'name' attribute, in the same namespace, then it is a schema definition error.

Here is an example of a format definition:

```
<xs:schema ...>
  <xs:annotation>
    <xs:appinfo source="http://www.ogf.org/dfdl/">
      <dfdl:defineFormat name="myConfig" >
        <dfdl:format representation="text"
                     ref="textSpecialFormat1" />
      </dfdl:defineFormat>
    </xs:appinfo>
  </xs:annotation>
  ...
</xs:schema>
```

A dfdl:defineFormat serves only to supply a named definition for a format for reuse from other places. It does not cause any use of the representation properties it contains to describe any actual data.

7.2.1   Inheritance for dfdl:defineFormat

A dfdl:defineFormat declaration can inherit from another named format definition by use of the ref property of the dfdl:format annotation. This allows a single-inheritance hierarchy that reuses definitions. When one definition extends another in this way, any property definitions contained in its direct elements override those in any inherited definition.

Conceptually, the 'ref' inheritance chains can be *flattened* and removed by copying all inherited property bindings and then superseding those for which there is a local binding. Throughout this document we will assume inheritance is fully flattened. That is, all 'ref' inheritance is first removed by flattening before any other examination of properties occurs.

It is a schema definition error if use of the ref property results in a circular path.

Comment [SMH52]: Errata 2.1. **DONE**

7.2.2   Using/Referencing a Named Format Definition

See section 7.1.2 Ref Property

7.3   The dfdl:assert Statement Annotation Element

The dfdl:assert statement annotation element is used to assert truths about a DFDL model that are used only when parsing to ensure that the data are well-formed. These checks are separate from validation checking and are performed even when validation is off. This distinction is needed to ensure that switching validation off does not affect parsing.

Examples of dfdl:assert elements are below:

```
<dfdl:assert message="Value is not zero." test="{ ../x ne 0}" />

<dfdl:assert message="Precondition violation." >
        {../x le 0 and ../y ne "-->" and ..y ne "<!-" }
</dfdl:assert>


<dfdl:assert message="Postcondition violation."  testKind='expression'>
      {../x ne "'"}
</dfdl:assert>
```

7.3.1   Properties for dfdl:assert

Comment [m53]: **2.110**. *Section 7.3.1, 7.4.1.* Done

Comment [m54]: **3.23**. *Sections 7.3.1.* Done

A dfdl:assert annotation contains a test expression or a test pattern. The dfdl:assert is said to be successful if the test expression evaluates to true or the test pattern returns a non-zero length match, and unsuccessful if the test expression evaluates to false or the test pattern returns a zero length match. An unsuccessful dfdl:assert causes a processing error.

The dfdl:testKind attribute specifies whether an expression or pattern is used by the dfdl:assert. The expression or pattern can be expressed as an attribute or as a value.

```
<dfdl:assert  test="{test expression}" />

<dfdl:assert>
          {test expression}
</dfdl:assert>
```

It is a schema definition error if a property is specified in more than one form.

It is a schema definition error if both a test expression and a test pattern are specified.

A dfdl:assert can appear as an annotation on:

- an xs:element declaration (local or global)

- an xs:element reference

- an xs:group reference

- an xs:sequence

- an xs:choice

- an xs:simpleType definition (local or global)

If the resolved set of statement annotations for a schema component contains 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 xs:sequence constructs to control the timing of evaluation of statements more precisely.

Property Name	Description
testKind	Enum (optional)
	Valid values are 'expression', 'pattern'
	Default value is 'expression'
	Specifies whether a DFDL expression or DFDL regular expression is used in the dfdl:assert.
	Annotation: dfdl:assert
test	DFDL Expression
	Applies when dfdl:testKind is 'expression'
	A DFDL expression that evaluates to true or false. If the expression evaluates to true then parsing continues. If the expression evaluates to false then a processing error is raised.
	Any element referred to by the expression must have already been processed or is a descendent of this element.
	The expression must have been evaluated by the time this element and it descendents have been processed.

	If a processing error occurs during the evaluation of the test expression then the dfdl:assert also fails. It is a schema definition error if dfdl:test is the empty string and the value is not specified and dfdl:testKind is 'expression' or not specified. Annotation: dfdl:assert
testPattern	DFDL Regular Expression Applies when dfdl:testKind is 'pattern' A DFDL regular expression that is applied against the data stream starting at the data position corresponding to the beginning of the representation. Consequently the framing (including any initiator) is visible to the pattern.at the start of the component on which the dfdl:assert is positioned. If the length of the match is zero then the dfdl:assert evaluates to false and a processing error is raised. If the length of the match is non-zero then the dfdl:assert evaluates to true. If a processing error occurs during the evaluation of the test regular expression then the dfdl:assert also fails. It is a schema definition error if dfdl:testPattern is the empty string and the value is not specified and dfdl:testKind is 'pattern'. In order for a testPattern to be used, the data subject to the pattern must be scannable using a DFDL regular expression otherwise the results are not predictable. It is a schema definition error if there is no value for the dfdl:encoding property in scope. It is a schema definition error if dfdl:leadingSkip is other than 0. It is a schema definition error if the dfdl:alignment is not 1 or 'implicit' Annotation: dfdl:assert
Message	String Defines text to be used as a diagnostic code or for use in an error message. The DFDL specification does not specify how a DFDL processor uses this message text. Annotation: dfdl:assert
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. If an error occurs while evaluating the test expression, a processing error occurs, not a recoverable error. Recoverable errors do not cause backtracking like processing errors. Annotation: dfdl:assert

Comment [SMH55]: The criteria are different for delimited and pattern.

DONE

Comment [SMH56]: 3.9 update. DONE

Comment [SMH57]: Errata 3.9 lists restrictions that apply to patterns. Might be ok to refer to section 12.3.5.1 to avoid duplication?

DONE

Comment [m58]: Errata 3.4

DONE

Table 9 dfdl:assert properties

7.4   The dfdl:discriminator Statement Annotation Element

DFDL discriminators are used to resolve points of uncertainty that cannot be resolved by speculative parsing. They can also be used to force a resolution earlier during the parsing of a group so that subsequent parsing errors are treated as processing errors of a known component rather than a failure to find a component.

A discriminator determines the existence or non-existence of a component. If the discriminator is successful then the component is known to exist and any subsequent errors will not cause backtracking at points of uncertainty. If a discriminator is unsuccessful then the component is known not to exist and backtracking occurs immediately.

If the complex type of an element contains a sequence group as its content model then if the sequence group is known not to exist, then the element is known not to exist.

Examples of dfdl:discriminator annotation are below :

```
<dfdl:discriminator>
   { ../recType eq 0 }
</dfdl:discriminator>

<dfdl:discriminator test="{ ../recType eq 0}" />
```

When the discriminator's expression evaluates to "false", then it causes a processing error, and the discriminator is said to fail.

7.4.1   Properties for dfdl:discriminator

A DFDL discriminator contains a test expression that is an expression that evaluates to true or false. The discriminator is said to be successful if the test evaluates to true and unsuccessful (or fails) if the test evaluates to false.

The dfdl:testKind attribute specifies whether an expression or pattern is used by the dfdl:discriminator. The expression or pattern can be expressed as an attribute or as a value.

```
<dfdl:discriminator  test="{test expression}" />

<dfdl:discriminator>
    { test expression }
</dfdl:discriminator>
```

It is a schema definition error if a property is specified in more than one form.

It is a schema definition error if both a test expression and a test pattern are specified.

A dfdl:discriminator can be an annotation on

- an xs:element declaration (local or global)
- an xs:element reference
- an xs:group reference
- an xs:sequence
- an xs:choice
- an xs:simpleType definition (local or global)

The resolved set of statement annotations for a schema component can contain only a single dfdl:discriminator or one or more dfdl:asserts, but not both. To clarify: dfdl:assert annotations and

dfdl:discriminator annotations are exclusive of each other. It is a schema definition error otherwise.

Property Name	Description
testKind	Enum Valid values are 'expression', 'pattern' Default value is 'expression' Specifies whether a DFDL expression or DFDL regular expression is used in the dfdl:discriminator . Annotation: dfdl:discriminator
test	DFDL Expression Applies when dfdl:testKind is 'expression' A DFDL expression that evaluates to true or false. If the expression evaluates to true then the discriminator succeeds and parsing continues. If the expression evaluates to false then the discriminator fails and a processing error is raised. If a processing error occurs during the evaluation of the test expression then the discriminator also fails. Any element referred to by the expression must have already been processed or is a descendent of this element. The expression must have been evaluated by the time this element and it descendents have been processed or when a processing error occurs when processing this element or its descendents. It is a schema definition error if dfdl:test is the empty string and the value is not specified and dfdl:testKind is 'expression' or not specified Annotation: dfdl:discriminator
testPattern	DFDL Regular Expression Applies when dfdl:testKind is 'pattern' A DFDL regular expression that is applied against the data stream starting at the data position corresponding to the beginning of the representation. Consequently the framing (including any initiator) is visible to the pattern.at the start of the component on which the dfdl:discriminator is positioned. If the length of the match is zero then the dfdl:discriminator evaluates to false and a processing error is raised. If the length of the match is non-zero then the dfdl:discriminator evaluates to true. It is a schema definition error if dfdl:testPattern is the empty string and the value is not specified and dfdl:testKind is 'pattern'. In order for a testPattern to be used, the data subject to the pattern must be scannable using a DFDL regular expression otherwise the results are not predictable. It is a schema definition error if there is no value for the dfdl:encoding property in scope. It is a schema definition error if dfdl:leadingSkip is other than 0.

Comment [SMH64]: The criteria are different for delimited and pattern.

DONE

Comment [SMH65]: 3.9 update. DONE

	It is a schema definition error if the dfdl:alignment is not 1 or 'implicit'
	Annotation: dfdl:discriminator
message	String
	Defines text to be used as a diagnostic code or for use in an error message. The DFDL specification does not specify how a DFDL processor uses this message text.
	Annotation: dfdl:discriminator

Comment [SMH66]: Need to use same words on scannability as dfdl:assert.

DONE

Comment [SMH67]: Errata 3.9 lists restrictions that apply to patterns. Might be ok to refer to section 12.3.5.1 to avoid duplication?

DONE

Table 10 dfdl:discriminator properties

```
<xs:sequence>
  <xs:choice>
    <xs:element  name='branchSimple' >
      <xs:annotation>
        <xs:appinfo source="http://www.ogf.org/dfdl/">
          <dfdl:discriminator test='{. eq "a"}'         />
        </xs:appinfo>
      </xs:annotation>
    </xs:element>

    <xs:element name='branchComplex' >
      <xs:annotation>
        <xs:appinfo source="http://www.ogf.org/dfdl/">
          <dfdl:discriminator test='{./identifier eq "b"}' />
        </xs:appinfo>
      </xs:annotation>
      <xs:complexType >
        <xs:sequence>
          <xs:element name='identifier'  />
          ...
        </xs:sequence>
      </xs:complexType>
    </xs:element>

    <xs:element name='branchNestedComplex' >
      <xs:annotation>
       <xs:appinfo source="http://www.ogf.org/dfdl/">
          <dfdl:discriminator test='{./Header/identifier eq "c"}'/>
        </xs:appinfo>
      </xs:annotation>
      <xs:complexType >
        <xs:sequence>
          <xs:element name='Header'  />
            <xs:complexType >
              <xs:sequence>
                <xs:element name='identifier'  />
                ...
              </xs:sequence>
            </xs:complexType>
          </xs:element>
        </xs:sequence>
      </xs:complexType>
    </xs:element>
```

```
    </xs:choice>
  </xs:sequence>
```

7.5   The dfdl:defineEscapeScheme Defining Annotation Element

One or more dfdl:defineEscapeScheme annotation elements can appear within the annotation children of the xs:schema. The dfdl:defineEscapeScheme elements may only appear as annotation children of the xs:schema.

The order of their appearance does not matter, nor does their position relative to other annotation or non-annotation children of the xs:schema.

Each dfdl:defineEscapeScheme has a required name attribute and a required dfdl:escapeScheme child element.

The construct creates a named escape scheme definition. The value of the name attribute is of XML type NCName. The name will become a member of the schema's target namespace. These names must be unique within the namespace among escape schemes.

If multiple dfdl:defineEscapeScheme definitions have the same 'name' attribute, in the same namespace, then it is a schema definition error.

Each dfdl:defineEscapeScheme annotation element contains a dfdl:defineEscapeScheme annotation element as detailed below.

Here is an example of an escapeScheme definition:

```
<xs:schema ...>
  <xs:annotation>
    <xs:appinfo source="http://www.ogf.org/dfdl/">
      <dfdl:defineEscapeScheme name="myEscapeScheme">
        ...
        <dfdl:escapeScheme escapeKind="escapeCharacter"
                           escapeCharacter='/' />
        ...
      </dfdl:defineEscapeScheme>
    </xs:appinfo>
  </xs:annotation>
  ...
</xs:schema>
```

A dfdl:defineEscapeScheme serves only to supply a named definition for an escapeScheme for reuse from other places. It does not cause any use of the representation properties it contains to describe any actual data.

7.5.1   Using/Referencing a Named escapeScheme Definition

A named, reusable, escape scheme is used by referring to its name from an escapeSchemeRef property on an element. For example:

```
<xs:element name="foo" type="xs:string" >
  <xs:annotation><xs:appinfo source="http://www.ogf.org/dfdl/">
    <dfdl:element representation="text"
                  escapeSchemeRef="myEscapeScheme"/>
  </xs:appinfo></xs:annotation>
</xs:element>
```

7.6   The dfdl:escapeScheme Annotation Element

The escapeScheme annotation is used within a dfdl:defineEscapeScheme annotation to group the properties of an escape scheme and allows a common set of attributes to be defined that can be reused.

An escape scheme defines the properties that describe the text escaping rules in force when data such as text delimiters are present in the data. There are two variants on such schemes,

- The use of a single escape character to cause the next character to be interpreted literally. The escape character itself is escaped by the escape escape character.

- The use of a pair of escape strings to cause the enclosed group of characters to be interpreted literally. The ending escape string is escaped by the escape escape character.

On parsing, the escape scheme is applied after padding characters are trimmed and on unparsing before padding characters are added.

DFDL does not perform any substitutions for ampersand notations like <.

The syntax of escapeScheme is defined in Section 13.2.1

The dfdl:escapeScheme Properties.

7.7   The dfdl:defineVariable Annotation Element

Variables provide a means for communication within a set of DFDL schema. They are defined as top-level elements in a schema and therefore have global scope. .

A new variable is introduced using dfdl:defineVariable:

```
<dfdl:defineVariable
      name = NCName
      type? = QName
      defaultValue? = logical value or dfdl expression
      external? = 'false' | 'true' >
  <!-- Contains: logical value or dfdl expression (default value) -->
</dfdl:defineVariable>
```

The name of a newly defined variable is placed into the target namespace of the schema containing the annotation. Variable names are distinct from format and escape scheme names and so cannot conflict with them. A variable can have any type from the DFDL subset of XML schema simple types. If no type is specified, the type is xs:string.

The defaultValue is optional. This is a literal value or an expression which evaluates to a constant, and it can be specified as an attribute or as the element value. If specified the default value must match the type of the variable (otherwise it is a schema definition error).

Note that the syntax supports both a defaultValue attribute and the 'defaultValue' being specified by the element value. Only one or the other may be present. (Schema definition error otherwise.)

Note the value of the name attribute is an NCName. The name of a variable is defined in the target namespace of the schema containing the definition. If multiple dfdl:defineVariable definitions have the same 'name' attribute in the same namespace then it is a schema definition error.

A default instance of the variable is created (with global scope). Further instances of the variable may subsequently be created on schema elements. If the variable has a default value, this will used as the default value for any instances of the variable (unless overridden when the instance is created).

The external attribute is optional. If not specified it takes the default value 'false'. If true the value may be provided by the DFDL processor and this external value will be used as the global default

---

Comment [m68]: (DONE)
Decided not to mention these functions here

new functions to do exactly this? String2StringLiteral and StringLiteral to String? (Errata forthcoming.)

Comment [m69]: **3.19**. *Sections 7.7*. DONE

Comment [SMH70]: Errata 2.12: Clause added. **DONE**

Comment [m71]: Misleading wording removed. All variable names, like all global definition names in XSD, are global to the entire schema consisting of all included/imported schema documents.That's what namespaces are about. We don't need to discuss the scope of the names of things. Only of the instances due to newVariableInstance.

DONE

value (overriding any defaultValue specified on the dfdl:defineVariable). The mechanism by which the processor provides this value is unspecified and implementation specific.

There is no required order between dfdl:defineVariable and other schema level defining annotations or a dfdl:format annotation that may refer to the variable.

A defaultValue expression is evaluated before processing the data stream begins.

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.

It is a schema definition error if the type of the variable is a user-defined simple type restriction.

7.7.1 Examples

```
 <dfdl:defineVariable name="EDIFACT_DS" type="xs:string"
                      defaultValue="," />

<dfdl:defineVariable name="codepage" type="xs:string"
                     external="true">utf-8</dfdl:defineVariable>
```

7.7.2 Predefined Variables

The following variables are predefined

Name	Namespace URI	Type	Default value	External
Encoding	http://www.ogf.org/dfdl/dfdl-1.0/	xs:string	'UTF-8'	true
byteOrder	http://www.ogf.org/dfdl/dfdl-1.0/	xs:string	'bigEndian'	true
binaryFloatRep	http://www.ogf.org/dfdl/dfdl-1.0/	xs:string	'ieee'	true
outputNewLine	http://www.ogf.org/dfdl/dfdl-1.0/	xs:string	'%LF;'	true

Table 11 Pre-defined variables

These variables are expected to be commonly set externally so are predefined for convenience.

```
        <xs:element name="title" type="xs:string">
          <xs:annotation>
            <xs:appinfo source="http://www.ogf.org/dfdl/">
              <dfdl:element encoding="{$dfdl:encoding}" />
            </xs:appinfo>
          </xs:annotation>
        </xs:element>
```

7.8   The dfdl:newVariableInstance Statement Annotation Element

Scoped instances of defined variables are created using dfdl:newVariableInstance:

```
<dfdl:newVariableInstance
      ref = QName
      defaultValue? = logical value or dfdl expression >
  <!-- Contains: logical value or dfdl expression (value) -->
</dfdl:newVariableInstance>
```

Comment [m72]: **3.20**. *Sections 7.8*. Additions and clarifications for the newVariableInstance annotation. DONE

Since an initial instance is created when the variable is defined, the use of dfdl:newVariableInstance is optional. It would be used if an instance with restricted scope is needed.

The dfdl:newVariableInstance annotation can be used on a group reference, sequence or choice only. It is a schema definition error otherwise.

The scope of the instance of a variable is the *dynamic scope* of the schema component and its content model and so is inherited by any contained constructs or construct references.

The ref attribute is a QName. That is, it may be qualified with a namespace prefix.

An optional defaultValue *for the instance* may be specified. It can be specified as an attribute or as the element value. The expression must not contain forward references to elements which have not yet been processed nor to the current component. If specified the default value must match the type of the variable as specified by dfdl:defineVariable. If the instance is not assigned a new default value then it will inherit the default value specified by dfdl:defineVariable or externally provided by the DFDL processor. If a default value is not specified (and has not been specified by dfdl:defineVariable) then the value of this instance is undefined until explicitly set (using dfdl:setVariable).

If a default value is specified this initial value of the instance will be set when the instance is created. The value will override any (global) default value which was specified by dfdl:defineVariable or which was provided externally to the DFDL processor. A variable instance with a valid value (specified or default) can be referenced anywhere within the scope of the element on which the instance was created.

Note that the syntax supports both a defaultValue attribute and the 'defaultValue' being specified by the element value. Only one or the other may be present. (Schema definition error otherwise.)

The resolved set of annotations for a component may contain multiple dfdl: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.

There is no short form syntax for creating variable instances.

7.8.1   Examples

```
<dfdl:newVariableInstance ref="EDIFACT_DS" defaultValue=","/>

<dfdl:newVariableInstance ref="lengthUnitBits">
    { if dfdl:property("lengthUnits")eq "bits" then 1 else 8 }
</dfdl:newVariableInstance>
```

7.9   The dfdl:setVariable Statement Annotation Element

Variable instances get their values either by default, by external definition, or by subsequent assignment using the dfdl:setVariable statement annotation.

```
<dfdl:setVariable
     ref = QName
     value? = logical value or dfdl expression
 <!-- Contains: logical value or dfdl expression (value) -->
</dfdl:setVariable>
```

The dfdl:setVariable annotation can be used on a simpleType, group reference, sequence or choice. It may be used on an element or element reference only if the element is of simple type. It is a schema definition error if dfdl:setVariable appears on an element of complex type, or an element reference to an element of complex type.

The ref attribute is a QName. That is, it may be qualified with a namespace prefix.

Comment [m73]: **3.21**. *Sections 7.9*. Additions and clarifications for the setVariable annotation. (DONE)

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

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. (DONE)

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

The syntax supports both a value attribute and the 'value' being specified by the element value. Only one or the other may be present. (Schema definition error otherwise.)

The value must match the type of the variable as specified by dfdl:defineVariable.

A dfdl:setVariable value expression may refer to the value of this element using a relative path value ".". Use of relative path expressions is recommended wherever possible as this will allow the behavior of the parser to be more effectively scoped. However this practice is not enforced and there may be situations in which use of an absolute path is in fact necessary.

The declaration of a variable must be in scope at the point of the assignment, and at the point of reference.

In normal processing, the value of an instance can only be set once using dfdl:setVariable. Attempting to set the value of the variable instance for a second time is a schema definition error. In addition, if a reference to the variable's value has already occurred and returned a default or an externally supplied value, then no assignment (even a first one) can occur. An exception to this behavior occurs whenever the DFDL processor backtracks because it is processing multiple arms of a choice or as a result of speculative parsing. In this case the variable state is also rewound.

A dfdl:setVariable will override any default value specified on either dfdl:defineVariable or dfdl:newVariableInstance, or externally.

The resolved set of annotations for an annotation point may contain multiple dfdl:setVariable statements. They must all be for unique variables and 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.

There is no short form syntax for variable assignment.

7.9.1   Examples

```
<xs:element name="ds" type="xs:string">
   <xs:annotation>< xs:appinfo source="http://www.ogf.org/dfdl/">
      <dfdl:setVariable ref="EDI:EDIFACT_DS" value="{.}" />
      <dfdl:setVariable ref="delta"> {.} </dfdl:setVariable>
   </xs:appinfo></xs:annotation>
</xs:element>
```

In the above example, the element named "ds" contains the string to be used as the EDI:EDIFACT_DS delimiter at other places in the data, so the above defines the value of the EDI:EDIFACT_DS variable to take on the value of this element.

8.   Property Scoping Rules

This section describes the rules that govern the scope over which DFDL representation properties apply

The scope of the representational properties on each of the component format annotations is given in **Table 12 DFDL annotation scoping**

Annotation Point	Property Scope
Schema declaration	dfdl:format representation properties apply *lexically* as default properties over all components in the schema
Element declaration	dfdl:element properties apply locally
Element reference	dfdl:element properties apply locally
Simple type definition	dfdl:simpleType properties apply locally
Sequence	dfdl:sequence properties apply locally
Choice	dfdl:choice properties apply locally
Group reference	dfdl:group properties apply locally

Table 12 DFDL annotation scoping

Note: This table lists DFDL annotations on schema components. DFDL annotations can also be placed on other DFDL annotations, such as a dfdl:format within a dfdl:defineFormat, to provide a named reusable format definition. In this case the annotation applies only where the named format is referenced.

DFDL representation properties explicitly defined on annotations, other than a dfdl:format on an xs:schema declaration, apply locally to that component only. The explicitly defined properties are the combination of any defined locally on the annotation and any defined on the dfdl:defineFormat annotation referenced by a local dfdl:ref property. When a property is defined both locally and on the dfdl:defineFormat, the locally defined property takes precedence.

The dfdl:format annotation on the top level xs:schema declaration provides defaults for the DFDL representation properties at every DFDL-annotatable component contained in the schema document. They do not apply to any components in any included or imported schema document (these may have their own defaults).

8.1   Providing Defaults for DFDL properties

A dfdl:format annotation on the top level xs:schema declaration may provide defaults for some or all the DFDL representation properties at every annotation point within the schema document. The default properties may be specified in attribute or element form. (Short form is not allowed on the xs:schema element.)

The dfdl:ref property is not a representation property so no default can be set.

The dfdl:escapeSchemeRef property provides a default reference to a dfdl:defineEscapeScheme, the properties of dfdl:escapeScheme are not defaulted individually.

DFDL representation properties defined explicitly on a component apply only to that component and override the default value of that property provided by a default format specified by an xs:schema dfdl:format annotation.

The example below demonstrates the overriding of a format encoding property. The `'ASCII'` dfdl:format encoding is the default value for the `title` element, but then it is overridden by the locally defined `utf-8` format encoding, which takes precedence.

```
<xs:schema>
```

```
  <xs:annotation>
    <xs:appinfo source="http://www.ogf.org/dfdl/">
      <dfdl:format encoding="ASCII" />
    </xs:appinfo>
  </xs:annotation>

  <xs:element name="book">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="title" type="xs:string">
          <xs:annotation>
            <xs:appinfo source="http://www.ogf.org/dfdl/">
              <dfdl:element encoding="utf-8" />
            </xs:appinfo>
          </xs:annotation>
        </xs:element>
        <xs:element name="pages" type="xs:int"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

8.2   Combining DFDL Representation Properties from a dfdl:defineFormat

The DFDL representation properties contained in a referenced dfdl:defineFormat are combined
with any DFDL representation properties defined locally on a construct as if they had been
defined locally. If the same property is defined locally in and in the referenced dfdl:defineFormat
then the local property takes precedence. The combined set of explicit DFDL properties has
precedence over any defaults set by a dfdl:format on the xs:schema.

```
<xs:schema>
  <xs:annotation>
    <xs:appinfo source="http://www.ogf.org/dfdl/">
      <dfdl:defineFormat name='myFormat'>
        <dfdl:format encoding="ASCII" />
      </dfdl:defineFormat>
    </xs:appinfo>
  </xs:annotation>

  <xs:element name="book">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="title" type="xs:string">
          <xs:annotation>
            <xs:appinfo source="http://www.ogf.org/dfdl/">
              <dfdl:element ref='myFormat' encoding="UTF-8" />
            </xs:appinfo>
          </xs:annotation>
        </xs:element>
        <xs:element name="pages" type="xs:int"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

The example above demonstrates the overriding of an encoding property. The 'ASCII' format
encoding from the 'myFormat' is overridden by the UTF-8 format encoding, which as a locally
defined property takes precedence.

8.3   Combining DFDL Properties from References

The DFDL properties from the following types of reference are combined using the rules below:

- An xs:element and its referenced xs:simpleType restriction,

- An xs:element reference and its referenced global xs:element

- An xs:group reference and an xs:sequence or xs:choice in its referenced global xs:group

- An xs:simpleType restriction and its base xs:simpleType restriction

Rules

1. Create an empty working set of "explicit" properties. Create an empty working set of "default" properties.

2. Move to the innermost schema component in the chain of references.

3. Assemble its applicable "explicit" properties from its local dfdl:ref (if present) and its local properties (if present), the latter overriding the former (that is, local wins over referenced).

4. Combine these with the current working set of "explicit" properties. It is a schema definition error if the same property appears twice. The result is a new working set of "explicit" properties.

5. Obtain applicable "default" properties from a dfdl:format annotation on the xs:schema that contains the component (if such annotation is present). Combine these with the current working set of "default" properties, the latter overriding the former (that is, inner wins). Result is a new working set of "default" properties.

6. Move to the schema component that references the current component, and repeat starting at step 3. If there is no referencing component, carry out step 5 and then go to step 7.

7. Combine the resultant sets of properties. The "explicit" properties take priority, "defaults" only used when no "explicit" is present. It is a schema definition error if a required property is in neither the "explicit" nor the "default" working sets.

"Applicable" properties are all the DFDL properties that apply to that schema component. For example all the DFDL properties that apply to a particular xs:simpleType (as defined by section 13).

```
<xs:simpleType name="newType">
  <xs:annotation>
    <xs:appinfo source="http://www.ogf.org/dfdl/">
      <dfdl:simpleType alignment="16"/>
    </xs:appinfo>
  </xs:annotation>
  <xs:restriction base="xs:integer">
    <xs:maxInclusive value="10"/>
  </xs:restriction>
</xs:simpleType>

<xs:element name="testElement1" type="newType">
  <xs:annotation>
    <xs:appinfo source="http://www.ogf.org/dfdl/">
      <dfdl:element representation="binary"/>
    </xs:appinfo>
  </xs:annotation>
</xs:element>
```

Comment [m74]: DONE – fixed up broken numbering/indenting.

Comment [SMH75]: I think should say 'to a particular xs:simpleType'.
(DONE)

The locally defined dfdl:alignment property with value '16' from the xs:simpleType 'newType' is combined with the locally defined dfdl:representation property with value 'binary' and applied to element 'testElement1',

```
<xs:simpleType name="otherNewType">
  <xs:annotation>
    <xs:appinfo source="http://www.ogf.org/dfdl/">
      <dfdl:simpleType alignment="64"/>
    </xs:appinfo>
  </xs:annotation>
  <xs:restriction base="newType">
    <xs:maxInclusive value="5"/>
  </xs:restriction>
</xs:simpleType>

<xs:simpleType name="newType">
  <xs:annotation>
    <xs:appinfo source="http://www.ogf.org/dfdl/">
      <dfdl:simpleType representation='binary'/>
    </xs:appinfo>
  </xs:annotation>
  <xs:restriction base="xs:int">
    <xs:maxInclusive value="10"/>
  </xs:restriction>
</xs:simpleType>
```

The locally defined dfdl:representation property with value 'binary' is combined with the locally defined dfdl:alignment property with value '64' from the xs:simpleType restriction 'otherNewType'.

```
<xs:sequence>
  <xs:element ref="testElement1">
    <xs:annotation>
      <xs:appinfo source="http://www.ogf.org/dfdl/">
        <dfdl:element binaryNumberRep ="binary"/>
      </xs:appinfo>
    </xs:annotation>
  </xs:element>
</xs:sequence>

<xs:element name="testElement1" type="newType">
  <xs:annotation>
    <xs:appinfo source="http://www.ogf.org/dfdl/">
      <dfdl:element representation="binary"/>
    </xs:appinfo>
  </xs:annotation>
</xs:element>

<xs:simpleType name="newType">
  <xs:annotation>
    <xs:appinfo source="http://www.ogf.org/dfdl/">
      <dfdl:simpleType alignment="16"/>
    </xs:appinfo>
  </xs:annotation>
  <xs:restriction base="xs:int">
    <xs:maxInclusive value="10"/>
  </xs:restriction>
</xs:simpleType>
```

The locally defined dfdl:alignment property with value '16' from the xs:simpleType 'newType' is combined with the locally defined dfdl:representation property with value 'binary' and locally defined dfdl:binaryNumberRep with a value of 'binary'

```
<!-- SCHEMA1 -->
<xs:schema targetNamespace="" xmlns:tns1="http://tns1">

  <xs:annotation>
    <xs:appinfo source="http://www.ogf.org/dfdl/">
      <dfdl:format encoding="ASCII" byteOrder="littleEndian"
                initiator="" terminator=""
                sequenceKind="ordered"  />
    </xs:appinfo>
  </xs:annotation>

  <xsd:import namespace="http://tns2" schemaLocation="SCHEMA2.xsd"/>

  <xs:element name="book">
    <xs:complexType>
      <xs:group ref="tns2:ggrp1" dfdl:separator=","></xs:group>
    </xs:complexType>
  </xs:element>

</xs:schema>
```

<div style="float:right">**Comment [SMH76]:** Remove inputValueCalc, as not allowed in dfdl:format

DONE</div>

```
<!-- SCHEMA2 -->
<xs:schema targetNamespace="" xmlns:tns2="http://tns2">

  <xs:annotation>
    <xs:appinfo source="http://www.ogf.org/dfdl/">
      <dfdl:format encoding="UTF-8" byteOrder="littleEndian"
                initiator=""
                sequenceKind="ordered"  />
    </xs:appinfo>
  </xs:annotation>

  <xs:group name="ggrp1" >
    <xs:sequence dfdl:separatorPosition="infix" >
      <xs:element name="customer" type="xs:string"
              dfdl:length="8" dfdl:lengthKind="explicit" />
    </xs:sequence>
  </xs:group>
</xs:schema>
```

<div style="float:right">**Comment [SMH77]:** I think we can drop this property altogether

DONE</div>

The DFDL properties applied to the xs:sequence in xs:group "ggrp1" in SCHEMA2 when referenced from the group reference in SCHEMA1 are

1. dfdl:separator="," from the group reference in SCHEMA1

2. dfdl:separatorPosition="infix" from the group declaration in SCHEMA2

3. dfdl:encoding="UTF-8", dfdl:initiator="" from the default dfdl:format annotation in SCHEMA2

4. dfdl:terminator ="" from the default dfdl:format annotation in SCHEMA1

<div style="float:right">**Comment [SMH78]:** We are talking about a sequence here, so I think we can drop this property altogether

DONE

Comment [m79]: Removed inputValueCalc since Errata 3.2 doesn't allow it here anymore.</div>

9.   DFDL Processing Introduction

A *DFDL Parser* is an application or code library that takes as input:

- A DFDL annotated schema
- A data stream.

It is able to use the DFDL schema description to interpret the data stream and realize the DFDL Information Model. This information set could then be written out (for example it could be realized as an XML text string) or it could be accessed by an application through an API (for example, a DOM-like tree could be created in memory for access by applications).

Symmetrically, there is a notion of a *DFDL Unparser.* The unparser works from an instance of the DFDL Information Model, a DFDL annotated schema and writes out to a target data stream in the appropriate representation formats.

Often both parser and unparser would be implemented in the same body of software and so we do not always distinguish them. Collectively they are called a *DFDL Processor*. The parser and unparser may, of course, be different bodies of software. Conforming DFDL processors may implement only a parser, because the unparser is an optional feature of DFDL.

9.1   Parser Overview

The DFDL logical parser is a recursive-descent parser [RDP] having guided, but potentially unbounded look ahead that is used to resolve *points of uncertainty*. (See 1.1) A DFDL parser reads a specification (the DFDL schema) and it recursively walks down and up the schema as it processes the data. This is done in a manner consistent with the scoping of properties and variables described in Section 8 Property Scoping Rules.

> **Comment [SMH80]:** TODO: Broken link there is no 9.1.1 now. (OPEN)

The unbounded look ahead means that there are situations where the parser must speculatively attempt to parse data where the occurrence of a processing error causes the parser to suppress the error, back out and make another attempt.

Implementations of DFDL may provide control mechanisms for limiting the speculative search behavior of DFDL parsers. The nature of these mechanisms is beyond the scope of the DFDL specification which defines the behavior of conforming parsers only on correct data. That is, data that can be parsed without any effective processing errors.

The logical parser recursively descends the DFDL schema beginning with the element declaration specified (in an implementation specific manner, see Section 18) of the *distinguished root node* of the schema passed to the DFDL processor. Depending on the kind of schema construct that is encountered and the DFDL annotations on it, and the pre-existing context, the parser performs specific parsing operations on the data stream. These parsing operations typically recognize and consume data from the stream and construct values in the logical model. For values of complex types and for arrays, these logical model values may incorporate values created by recursive parsing.

DFDL Implementations are free to use whatever techniques for parsing they wish so long as the semantics are equivalent to that of the speculative recursive-descent logical parser described in this specification. It is required that implementations distinguish the various kinds of errors (schema definition error, processing error, etc.) no matter what time they are detected. Some implementations may not detect certain schema definition errors until data are being parsed; however, they must still distinguish schema definition errors (which indicate that the schema itself is not meaningful), from parsing errors (which indicate that the input data doesn't satisfy the requirements of the schema), or unparsing errors (which indicate that the infoset does not satisfy the requirements of the schema).

9.2   DFDL Data Syntax Grammar

Data in a format describable via a DFDL schema obeys the grammar given here. A given DFDL schema is read by the DFDL processor to provide specific meaning to the terminals and decisions in this grammar.

The bits of the data are divided into two broad categories:

1   Content

2   Framing

The content is the bits of data that are interpreted to compute a logical value.

Framing is the term we use to describe the delimiters, length fields, and other parts of the data stream which are present, and may be necessary to determine the length or position of the content of DFDL Infoset items.

Note that sometimes the framing is not strictly necessary for parsing, but adds useful redundancy to the data format, allowing corrupt data to be more robustly detected, and sometimes the framing adds human readability to the data format.

In the grammar tables below, the terminal symbols are shown in bold italic font.

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

Comment [SMH83]: Need this term so can talk in general terms in 13.15.
OK
(DONE)

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**

Table 13 DFDL Grammar Productions

XML Schema and DFDL properties are used to control constraints on the terminals of the above grammar, as well as repetition (the "*" operator), and alternatives (the "|" operator). For a given set of XML Schema and DFDL properties, and prior data, any terminal may be allowed to be length zero, to contain specific data, or to contain a variety of different admissible data.

Some definitions are needed to cover the range of representations that are possible in the data stream for an element. These definitions are with respect to the grammar above.

9.2.1   Nil Representation

An element occurrence has a *nil representation* if the element has XSDL nillable property 'true' and the occurrence either:

   a) conforms to the grammar for SimpleNilLiteralElementRep or
       ComplexNilLiteralElementRep. Specifically, the **NilElementInitiator** and
       NilElementTerminator regions must be conformant with dfdl:nilValueDelimiterPolicy. (If
       non-conformant it is not a processing error and the representation is not nil).
   b) conforms to the grammar for SimpleNormalRep and its value is **NilLogicalValue**.

The LeadingAlignment, TrailingAlignment, PrefixLength regions may be present.

9.2.2   Empty Representation

An element occurrence has an *empty representation* if the occurrence does not have a nil representation and it conforms to the grammar for SimpleEmptyElementRep or ComplexEmptyElementRep. Specifically, the **EmptyElementInitiator** and **EmptyElementTerminator** regions must be conformant with dfdl:emptyValueDelimiterPolicy and the occurrence's content in the data stream is of length zero. (If non-conformant it is not a

Comment [SMH84]: Should be bold italics.

DONE

Comment [SMH85]: Errata 3.7 adds BOM to this

DONE

Comment [SMH86]: Errata 2.55. **DONE**

Comment [SMH87]: Errata 2.13. **DONE**

Comment [SMH88]: Errata 3.10 add complex nils which changes this

DONE

Comment [SMH89]: Should be bold

DONE

Comment [SMH90]: Uncomfortable with the lack of symmetry – nil rep is in ComplexContent but not SimpleContent

Comment [m91]: I think there are a few changes to the grammar around this topic, so rather than fix just this, I'd like to look at a proposal to change the grammar quite a bit so that it is at least not misleading or in conflict with a grammar that could describe an implementation.

Comment [m92]: DONE (I withdraw the issue. NilRep doesn't need to be broken down further. Not at all sure that would be helpful.)

Comment [SMH93]: Errata 2.55. **DONE**

Comment [SMH94]: Errata 2.55 renames this. **DONE**

Comment [SMH95]: Errata 2.61 refactors this. **DONE**

Comment [m96]: DONE
Inserted this language. Seemed it was just left unsaid.

OK

Comment [m97]: DONE: Issue 140 material plugged in here.

some of this material moved to other sections such as Section 13.15 on Nils and Defaults.

Comment [m98]: DONE
Does this phrase need to be repeated for each of the Empty and Normal representation as well?

OK as is.

processing error and the representation is not empty). LeadingAlignment, TrailingAlignment, PrefixLength regions may be present.

The *empty representation* is special in DFDL, because when parsing it is this condition that can trigger the creation of a default value for an element occurrence. See Section (TBD: Cross Reference) below about default values.

9.2.3   Normal Representation

An element occurrence has a normal representation if the occurrence does not have the nil representation or the empty representation and it conforms to the grammar for SimpleNormalRep or ComplexNormalRep.

9.2.4   Absent Representation

An element occurrence has an *absent representation* if the occurrence does not have a nil or empty or normal representation, and it conforms to the grammar for AbsentElementRep. Specifically, the occurrence's representation in the data stream is of length zero. Consequently, the Initiator, Terminator, LeadingAlignment, TrailingAlignment, PrefixLength regions must not be present.

Example of an absent representation: During unparsing, if an optional element does not have an item in the infoset then nothing is output. However if a separator of an enclosing structure is subsequently output as the immediate next thing, then a subsequent parse of the element may return a representation of length zero. If this happens, and this zero-length representation does not conform to any of the nil representation, the empty representation, or the normal representation, then it is the absent representation, and it behaves *as if the element occurrence is 'missing'*. (The term 'missing' is defined below.)

The point of this term 'absent representation', is that often we know the location where an element or group's representation would be in the data based on the delimiters of an enclosing group. (An example: if there are adjacent delimiters of an enclosing sequence.) When this location in the data, which is of zero length, cannot be a nil, empty, or normal representation, then we say it has absent representation, or "the representation is absent".

9.2.5   Zero-length Representation

We use the term *zero-length representation* to describe the situations where any of the above representations turn out to be of length zero due to specific combinations of data type and format properties:

- The nil representation can be a zero-length representation if dfdl:nilValue="%ES;", and there is no framing or framing is suppressed by dfdl:nilValueDelimiterPolicy

- The empty representation can be a zero-length representation if there is no framing or framing is suppressed by dfdl:emptyValueDelimiterPolicy.

- The normal representation can be a zero-length representation if the type is xs:string or xs:hexBinary and there is no framing.

- The absent representation always has a zero-length representation.

If the nil representation may be zero-length, then the absent representation cannot occur because zero-length will be interpreted as nil representation.

If the nil representation may not be zero length, but the empty representation is zero-length, then the absent representation cannot occur because zero-length will be interpreted as the empty representation.

If the nil and empty representations can not be zero-length, but the normal representation MAY be zero length then the absent representation cannot occur because zero length will be interpreted as a normal representation.

Comment [SMH99]: Deleted the conditions for when defaults are applied as they were incomplete and the details are in 9.4. (DONE)

Comment [m100]: (OPEN) Update cross reference.

Comment [m101]: DONE Added this whole section on zero-length representation, as we use it in several places, but very much need this in the discussion of separator suppression.

But the definition really belongs here.

Comment [m102]: (DONE) For normal representation, add that there is also a complexType case where we can have zero-length representation (that is not the nillable case) – when all children are optional, or all but the first child is optional, and that child can have zero-length representation.

SMH: That's not normal representation, that'e empty representation.

Decided that this clarification isn't needed because it is handled elsewhere.

If the nil representation may not be zero-length, the empty representation is not zero-length, and the normal representation may not be zero-length, then a zero-length representation is the absent representation, or "is absent".

9.2.6   Missing

When parsing, an element occurrence is missing if it does not have nil, empty, or normal representations, or it has the absent representation.

When parsing, the term missing really covers two situations. Firstly it subsumes absent representation. Secondly it applies when an element does not have a representation at all in the data stream, that is, when we do not even have the constructs in the data stream to determine the representation of the element; hence, none of the concepts above apply. This will be made clearer in the examples below. If an element occurrence is missing when parsing, no item is ever added to the Infoset.

When unparsing, an element occurrence is missing if there is no item in the infoset. For a required element occurrence, it is this condition that can trigger the creation of a default value in the augmented infoset. See Section (TBD: Cross Reference) below about default values. For an optional element occurrence, no item is ever added to the augmented Infoset nor any representation ever output in the data stream.

9.2.7   Examples of Missing and Empty Representation

The following examples illustrate missing and empty representation.

```
<xs:sequence dfdl:separator="," dfdl:terminator="@"
            dfdl:separatorSuppressionPolicy="trailingEmpty">
    <xs:element name="A" type="xs:string"
                dfdl:lengthKind="delimited"/>
    <xs:element name="B" type="xs:string" minOccurs="0"
                dfdl:lengthKind="delimited"/>
    <xs:element name="C" type="xs:string" minOccurs="0"
                dfdl:lengthKind="delimited"/>
</xs:sequence>
```

In data stream `aaa,@` element B has the empty representation, and element C does not have a representation so is missing.

```
<xs:sequence dfdl:separator=","
            dfdl:separatorSuppressionPolicy="anyEmpty">
    <xs:element name="A" type="xs:string"
                dfdl:lengthKind="delimited" dfdl:initiator="A:"
                dfdl:emptyValueDelimiterPolicy=initiator"/>
    <xs:element name="B" type="xs:string" minOccurs="0"
                dfdl:lengthKind="delimited" dfdl:initiator="B:"
                dfdl:emptyValueDelimiterPolicy="initiator"/>
    <xs:element name="C" type="xs:string" minOccurs="0"
                dfdl:lengthKind="delimited" dfdl:initiator="C:"
                dfdl:emptyValueDelimiterPolicy=initiator"/>
</xs:sequence>
```

In data stream `A:aaaa,C:cccc` element B does not have a representation so is missing.

In data stream `A:aaaa,B:,C:cccc` element B has the empty representation.

Comment [m103]: DONE
Added this.

It seems like it would be better to keep absent and missing orthogonal rather than have them overlap like this.

Absent meaning we can determine the length, it is zero, and zero-length can't be nil, empty or normal. Adjacent delimiters being the most common example, but prefix length could also do it, etc.

Missing meaning we don't even have the ability to determine the length, as in we've already seen the group terminator, so the last several elements within the group don't have anything representing them at all.

SMH: I'm happy to keep missing as encompassing absent.Let's see how it unfolds.

Decided to keep as is.

Comment [SMH104]: Moved this here from later on. OK (DONE)

Comment [m105]: (OPEN)
Update cross reference.

Comment [SMH106]: Moved this here from later on. OK (DONE)

In the data stream `A:aaaa,,C:cccc` element B has the absent representation so is missing.

Note that round tripping, meaning the unparser writing out exactly what was parsed, is not always guaranteed. An empty string in the Infoset will be output as the empty representation, but if the element is nillable and empty string (%ES;) is a nil value and dfdl:nilValueDelimiterPolicy is the same as dfdl:emptyValueDelimiterPolicy, then when parsed the Infoset will contain nil. See Section (TBD: Cross reference) for more about round tripping ambiguities.

9.3 Parsing Algorithm

A DFDL parser proceeds by determining the existence of occurrences of schema components. It does this by examining the data and the schema, so as to:

a) Establish representation

b) Resolve points of uncertainty

These two activities are defined below. They are mutually recursive in the expected way as a DFDL schema is a recursive nest of schema components.

Establishing the representation of an occurrence of a schema component and resolving points of uncertainty involve the concepts of known-to-exist and known-not-to-exist.

9.3.1 Known-to-exist and Known-not-to-exist

9.3.1.1 Known-to-exist

An occurence of a schema component is said to be *known-to-exist* when any of these *positive discriminations* hold:

1. There is a dfdl:discriminator[9] applying to the component and its expression evaluates to true or regular expression pattern matches.

2. The component is a direct child of an xs:sequence or xs:choice with dfdl:initiatedContent 'yes' and an initiator defined for the component is found.

3. TBD: direct dispatch choices

4. The component contains an array element which has dfdl:occursCountKind='stopValue' and an occurrence of the dfdl:occursStopValue is parsed.

If none of those hold because they are not applicable then the occurrence is still known-to-exist if ALL of the following hold, and no processing error occurs during their determination:

1. There are dfdl:asserts with failureType 'processingError' on the component and all their expressions evaluate to true or their regular expression patterns match,

2. It has nil, empty, or normal representation

3. When it has normal representation, this of course implies that the content of the representation is convertible to the element type without error.

---

[9] DFDL discriminators are described in section: 7.4 The dfdl:discriminator Statement Annotation Element.

Comment [m107]: (OPEN)

Fill in cross reference.

Comment [m108]: (DONE)
Should we move this elsewhere?

Seems ok.

Comment [m109]: (DONE)
Better name for this than 'algorithm' perhaps? Behavior?

Decided to keep algorithm

Comment [m110]: (DONE)

2013-05-24 – will keep in mind for possible future clarification. But ok for now.

2013-05-09 – Still open. Model groups don't have nil, empty, normal, or absent reps as those are specifically about elements (above).

Still TBD.

I find this statement confusing

Do we just want to say that the length of a sequence or choice is determined bottom up by considering their content? … ah. I think that is also too simplistic.

SMH: It's trying to capture the case when a model group occurs in the content of a complex element

MJB: Issues: First off, a 'local' sequence or choice as opposed to a group reference to a global one? I don't see any difference in how they are treated for purposes of establishing representation. Are we after a sequence or choice that is not the immediate schema-child of a complexType def?

Second. What are we trying to convey by the phrase "delimiting regime of the parent". ... [1]

Comment [SMH111]: Could be on a child component but apply to this PoU.
OK (DONE)

Comment [SMH112]: TBD (OPEN)

Comment [m113]: OPEN
Stopvalue related.

Changed wording to be 'contains' to take perspective of containing model group.

Inserted this point based on it coming up below in section on PoU. ... [2]

Comment [m114]: (DONE)

I revised this from what it was in the Action 140 materials because I believe it was really wrong, or at least unclear about how discrimination interacts with the known-to-exist/not-exist decision.

... [3]

9.3.1.2 **Processing Error After Determining Known-to-exist**

Note that it is possible for an occurrence of a schema component to be known-to-exist due to a positive discrimination, but then subsequently a processing error occurs when evaluating a statement annotation such as a dfdl:assert or a dfdl:setVariable, or a processing error occurs when determining the representation, or in the case of normal representation and simpleType, when converting that representation's content into a value of the type. This processing error does not change the fact that the schema component was determined to be known-to-exist. This is important in the discussion of resolving *Points of Uncertainty* below.

9.3.1.3 **Known-not-to-exist**

An occurrence of a schema component is *known-not-to-exist* when any of these *negative discriminations* holds:

1. There is a dfdl:discriminator applying to the component and its expression evaluates to false or regular expression pattern fails to match, or a processing error occurs while processing the dfdl:discriminator.

2. The component is a direct child of an xs:sequence or xs:choice with dfdl:initiatedContent 'yes' and an initiator defined for the component is not found.

3. TBD: Direct dispatch negative case

If none of those hold because they are not applicable, then a schema component is known-not-to-exist when any of the following hold:

1. The occurrence is missing

2. There is a dfdl:assert with failureType 'processingError' on the component and its expression evaluates to false or its regular expression pattern fails to match, or a processing error occurs while processing the dfdl:assert.

3. A processing error occurs when parsing the component. Processing errors include, but are not limited to, inability to identify any of nil, empty, normal or absent representations, or failure to convert a value to the built-in logical type.

Note that validation errors or recoverable errors do not cause a component to be known-not-to-exist.

Note: based on the above, when processing a sequence for which a separator is defined, the presence of a match in the data for the separator is not sufficient to cause the parser to determine that an associated component is known-to-exist. See Section TBD about Separators.

9.3.2 **Establishing Representation**

Unless an element occurrence is known-not-to-exist, it must be established if it has the nil, empty, normal, or absent representation.

The first step is to see if the content is trivially of length zero. This is dfdl:lengthKind dependent.

- explicit => length is zero (either fixed or from expression evaluation)
- prefixed => length given by the prefix is zero
- implicit (simple) => length is zero[11]
- implicit (complex) => not possible.

---

[11] This is a corner case that only happens when type is xs:string or xs:hexBinary and the maxLength facet is 0. Such an element can only be of length 0.

Comment [m115]: (DONE)
Rewrote from what we had in action 140 to be fully symmetric with the known-to-exist that I also revised.

Ok in principle.

Comment [m116]: (OPEN) provide direct choice dispatch negative case.

Comment [m117]: DONE
Remove? (goes without saying).

SMH: Missing is sufficient

Comment [m118]: 2.105. *Section 9.1.1*. DONE

Comment [m119]: OPEN
Resolve cross reference.

Comment [m120]:
Feels very out of place. I think this statement belongs in the section on separator properties.

Ok with reference to the section on separators. DONE

Comment [m121]: (DONE)

Was redundant comment. Removed.

Comment [m122]:
DONE (removed)
These 3 paragraphs are entirely redundant with the definitions of the representations above, and with the sections on simple element and complex element below.

I suggest we solve this problem in the previous comment by just dropping these 3 paragraphs.

Comment [m123]: DONE
What's a type facet? Just give a such as ..example.
SMH: Its maxLength 0

Comment [m124]: DONE
Should this be content length?
SMH: Yes

- delimited => length is zero (delimiter is immediately encountered)
- pattern => pattern returns zero length match
- endOfParent => already positioned at parent's end so length is zero

9.3.2.1 Simple element

If the result is length zero as described above, the representation is then established by checking, in order for:

- nil representation (if %ES; is a literal nil value).

- empty representation.

- normal representation (xs:string or xs:hexBinary only)

- absent representation (if none of the prior representations apply).

If the result is not length zero, the representation is then established by checking, in order, for:

- nil representation (as a literal nil value)

- nil representation (as a logical nil value)

- normal representation

9.3.2.2 Complex element

If the result is length zero as described above, the representation is then established by checking for:

- nil representation (if %ES; is a literal nil value).[12]

To establish any other representations requires that the parser descends into the complex type for the element, and returns successfully (that is, no unsuppressed processing error occurs). If the result is zero bits consumed, the representation is then established by checking, in order, for:

- empty representation.

- absent representation (if none of the prior representations apply).

Otherwise the element has normal representation.

Note: The DFDL parser shall not recursively parse the schema components inside a complex element when it has already established that the element occurrence is missing[13].

9.3.3 Points of Uncertainty

A *point of uncertainty* occurs in the data stream when there is more than one schema component that might occur at that point. Points of uncertainty can be nested.

Any one of the following constructs is a potential point of uncertainty:

---

[12] It is a schema definition error if a complex element has XSDL nillable 'true' and dfdl:lengthKind 'implicit'.

[13] The rational for this is that otherwise this could give rise to misleading error messages where the parser reported that required child elements were missing required occurrences. (This is consistent with XML Schema validation, where if a required element is missing, it gets reported as such, and there is nothing reported about its children).

Comment [m125]: DONE

Do we need to clarify here about complex types, because this involves descending into the type (because there can be fixed-length things in there, perhaps inner things use the same delimiter as the enclosing element, etc.)

SMH: Using term 'trivially length zero' and 'delimiter immediately encountered'

Comment [m126]: DONE
If we want, we can rule this out and say that xs:string and xs:hexBinary can only have literalValue or literalCharacter nils. logicalValue would be an SDE.

I cannot see any loss of expressive power by doing this.

SMH: Agreed to do this. Will raise separate errata.

Comment [m127]: DONE
Question about the caveats in here that said type must be string or hexBinary and you are getting empty string or empty hexBinary.

Is this correct? I looked at textStandardZeroRep, and that doesn't allow ES, so zero length can't be the representation of zero except by a required element with a default value of 0, which is not this case.

Comment [m128]: OPEN
What about the stopValue case?

I.e., non-zero length, but matches the stopValue for OCK=stopValue.

From the perspective of the simple element that ends up being the stop value of an array, the representation is normal.

I suggest this detail just goes into a footnote. It is sort of distracting otherwise.

Comment [m129]: DONE

Moved here from below.

Rationale statement moved to a footnote. Removed redundant is missing or is absent or has no representation. "Is missing" is sufficient.

This statement, however, left me wondering "and how would the DFDL processor determine that?". We don't talk about determining that something is missing. We talk about determining if it is absent rep, which is one way it can be missing, but the other way we don't discuss algorithmically.

It also feels like it is in the wrong place.

Furthermore: this feels odd because the parser algorithm really dosen't determine that things are missing until after a bunch of conditional behaviors associated with complex types. So saying it doesn't recurse when it has established that it is missing seems wron... [4]

1. An xs:choice

2. All xs:elements in an unordered xs:sequence (dfdl:sequenceKind='unordered')

3. An optional xs:element

4. An array xs:element.

5. All xs:elements in an xs:sequence containing one or more floating xs:elements.

The parser resolves these points of uncertainty by way of a set of construct-specific rules given below along with determining whether schema components are known-to-exist or known-not-to-exist. For some of these constructs, whether there is an actual point of uncertainty depends on the representation of the constructs in the data.

An xs:choice is always a point of uncertainty. It is resolved by parsing each choice branch in schema definition order until one is known-to-exist. It is a processing error if none of the choice branches are known-to-exist.

An element in an unordered xs:sequence is always a point of uncertainty. It is resolved by parsing for the child components of the sequence in schema definition order at each point in the data stream where a component can exist until the required number of occurrences of each child component is known- to-exist or the sequence is terminated by delimiters or specified length.

An optional element point of uncertainty is resolved by parsing the element until it is either known-to-exist or known-not-to-exist. Whether an optional element is an actual point of uncertainty depends on property dfdl:occursCountKind as described in section 16.

For an array element, the point of uncertainty is resolved for each occurrence separately by parsing the occurrence until it is either known-to-exist or known-not-to-exist. Whether an individual occurrence is an actual point of uncertainty depends on property dfdl:occursCountKind as described in section 16. The array as a whole is known to exist if one of its occurrences exists[14].

An element in a sequence with one or more floating elements is always a point of uncertainty. It is resolved by parsing for the expected element at that point in the data stream. If the expected element is known-not-to-exist then an occurrence of each floating element is parsed in schema definition order. It is a processing error if neither the expected ordered component element nor any of the floating elements is known-to-exist.

9.3.3.1 Nested Points of Uncertainty

A point of uncertainty can be resolved because a schema component has been determined to be known-to-exist due to positive discrimination. In that case, if a subsequent processing error occurs when completing the parsing of that schema component this will cause the next enclosing schema component surrounding this point of uncertainty to be determined to be known-not-to-exist.

For example, when parsing an element occurrence for an array with a variable number of occurrences, a positive discrimination tells the parser that the currently-being-parsed occurrence is known-to-exist. If a subsequent processing error occurs while completing the parsing of this occurrence, then the entire array is then known-not-to-exist.

---

[14] If an array has occursCountKind='stopValue', then an occurrence of the stop value (only) resolves the array as known-to-exist, even though there will be no element occurrences corresponding to the array in the infoset.

Comment [m130]: (DONE) Should this just be a footnote?

SMH: I think we need the concept of potential and actual PoUs. Some constructs are always PoUs, some are potentially PoUs – specifically optional and arrays (not just variable arrays) Whether a potential PoU becomes an actual PoU depends on other properties. I've added occursCountKind..

OK, since there is a cross reference to section 16.4.

Comment [m131]: (OPEN) Augment description to include choices with direct dispatch.

Comment [SMH132]: OPEN: Cross reference

Comment [m133]: OPEN Why don't we just say it depends on occursCountKind and depending on that, it depends on the values of minOccurs and maxOccurs.

Comment [SMH134]: OPEN: Cross reference

Comment [SMH135]: OPEN Hmm – what about stopValue arrays? Is there the concept of array known-to-exist? Should this be deleted?

MJB: Interesting. Using a stopvalue one can have the concept of an array containing zero element occurrences but which is known-to-exist. However, this concept of an array with zero element occurrences isn't possible in our infoset. So, if a stopValue is found for an array, then the array is known to exist even though there are no occurrences. Strange, but seems true.

I added a footnote about this corner case.

I added an item in the section on known-to-exist above about this stopValue case. Perhaps the footnote should go on that instead of here.

Another example is a choice. If a discriminator resolves the choice point of uncertainty to the first of the choice's alternatives, a subsequent processing error causes the entire choice construct to be determined to be known-not-to-exist.

This will cause the next enclosing point of uncertainty to try the next possible alternative, or if there isn't one, will cause an unsuppressed processing error.

The behavior of a DFDL processor on an unsuppressed processing error is not specified, but it is allowable for implementations to abort further parsing.

9.4   Element Defaults

A DFDL processor can create element defaults in the Infoset for both simple and complex elements. This happens quite differently for parsing and unparsing as will be explained in this section.

9.4.1   Definition 'default value'

A simple element has a default value if any of these are true:

1. The XSDL default property exists. The default value is the property's value.

2. The XSDL fixed property exists. The default value is the proiperty's value.

3. The element has XSDL nillable='true' and dfdl:useNilForDefault ='yes'. The default value is the special value nil.

9.4.2   Element Defaults When Parsing

If *empty* representation is established when parsing, the possibility of applying an element default arises. Essentially, if a required occurrence of an element has empty representation, then an element default will be applied if present, though there are a couple of variations on this rule. Remember that in order to have established empty representation, the occurrence must be compliant with the dfdl:emptyValueDelimiterPolicy for the element, and for a complex element the parser must have descended into the type and returned with no unsuppressed processing error.

The rules for applying element defaults are not dependent on dfdl:occursCountKind. However, if a required occurrence does not produce an item in the Infoset after the rules have been applied, then whether it is a processing error or a validation error (if validation is enabled) *does* depend on dfdl:occursCountKind (see section 16).

There are three main cases to consider. In what follows the term 'string' encompasses both xs:string and xs:hexBinary as these are the two data types for which a zero length (empty) string is valid for the type.

9.4.2.1   Simple element (non-string)

Required occurrence: If the element has a default value then an item is added to the Infoset using the default value, otherwise nothing is added to the Infoset.

Optional occurrence: Nothing is added to the Infoset.

9.4.2.2   Simple element (string)

Required occurrence: If the element has a default value then an item is added to the infoset using the default value, otherwise an item is added to the Infoset using empty string as the value.

Optional occurrence: If dfdl:emptyValueDelimiterPolicy is not 'none'[15] then an item is added to the Infoset using empty string as the value, otherwise nothing is added to the Infoset.

Note: To prevent unwanted empty strings from being added to the Infoset, use xs:minLength > '0' and a dfdl:assert that uses the dfdl:checkConstraints() function, to raise a processing error.

9.4.2.3   Complex element

Required occurrence: An item is added to the Infoset.

Optional occurrence: If dfdl:emptyValueDelimiterPolicy is not 'none' then an item is added to the Infoset, otherwise nothing is added to the Infoset.

For both required and optional occurrences, the Infoset item may also have a child item.
   a) If the first child element of the complex type is a required simple element, then an empty string or default value will also be added to the Infoset.
   b) If the first child element of the complex type is a required complex element, then an item is added to the Infoset (which may itself have a child via a)

As an example, consider a sequence S0 with a separator that contains among other content an optional non-nillable non-initiated element E1 of complex type. The content of the type is a sequence S1 with a different separator and the first child is a required non-initiated element E2 of type xs:string. The dfdl:lengthKind of both E1 and E2 is 'delimited'. The representation of E1 has zero length, that is, the data contains adjacent S0 separators. On processing E1, the parser will establish a point of uncertainty and descend into E1's complex type and process E2. It scans for in-scope delimiters and immediately encounters S0 separator. E2 has the empty representation, so E1 is added to the Infoset along with a value of empty string for E2. All other content of S1 is missing, so the parser returns from the descent. E1 is therefore known-to-exist. Because the position in the data has not changed, E1 therefore has the empty representation. Because E1 is empty and optional it is not added to the Infoset, and the Infoset items for E1 and E2 are discarded.

9.4.3   Element Defaults When Unparsing

If an element is *missing* from the Infoset when unparsing, the possibility of applying an element default arises. Essentially if a required occurrence of an element is missing, then an element default will be applied if present, and the resulting item is added to the *augmented Infoset*.

The rules for applying element defaults are not dependent on dfdl:occursCountKind. However if a required occurrence does not produce an item in the augmented Infoset after the rules have been applied then whether it is a processing error or a validation error (if enabled) *is* dependent on dfdl:occursCountKind (see section 16).

There are two main cases to consider.

9.4.3.1   Simple element

Required occurrence: If an element has a default value then an item is added to the augmented Infoset using the default value, otherwise nothing is added.

Optional occurrence: Nothing is added to the augmented Infoset.

9.4.3.2   Complex element

Required occurrence: An item is added to the augmented Infoset as specified below.

---

[15] If other than 'none', either an initiator, terminator or both must have been found in the data stream.

Comment [m142]: (OPEN) update cross reference

Comment [m143]: DONE

Contradiction: says above that for unparsing this is INDEPENDENT of occursCountKind.

SMH: There is actually no contradiction, the difference is between applying the rules and the end result. Moved all the content about required occurrences to here which makes this clear.

Optional occurrence: Nothing is added to the augmented Infoset.

For a required occurrence, the unparser descends into the complex type:
o   For a sequence, each child element is examined in schema order and the rules for simple and complex elements applied (recursively). The lack of a default may give rise to a processing error, as described above.
o   For a choice, each branch is examined in schema order and the above rules applied recursively to the branch. The lack of a default may give rise to a processing error, as described above, and if so the error is suppressed and the next branch is tried, otherwise that branch is selected. It is a processing error if no choice branch is ultimately selected.

9.5   Evaluation Order for Statement Annotations

Given a component of a DFDL schema, there is a resolved set of annotations for it.

Of these, some are statement annotations and the order of their evaluation relative to the actual processing of the schema component itself (parsing or unparsing via its format annotations) 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)

9.5.1   Asserts and Discriminators with testKind 'expression'

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.

9.5.2   Discriminators with testKind 'expression'

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.

9.5.3   Elements and setVariable

Comment [m144]: DONE

Review: deleted this statement:

"If it has empty representation, then there are circumstances when an item is added to the Infoset, as described earlier. "

Action 140 decided this is the wrong behavior.

SMH: Actually it's the right behavior, see 9.4.1.3. But we don't need to restate it. Moved all the content about optional occurrences to the relevant places earlier on.

Comment [m145]: **3.25**. *Section 9*. Evaluation Order for Statement Annotations DONE

Comment [m146]: DONE

Action 140 related.

Moved from before the grammar to avoid breaking up the flow, as this section is very separable from the rest.

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.

10. Core Representation Properties and their Format Semantics

The next sections specify the core set of DFDL v1.0 properties that may be used in DFDL annotations in DFDL Schemas to describe data formats.

It is a schema definition error when a DFDL schema does *not* contain a definition for a representation property that is needed to interpret the data. For example, a DFDL schema containing any textual data must provide a definition of the character set encoding property (dfdl:encoding) for that textual data, and if it is not part of the format properties context for that data, then it is a schema definition error.

Furthermore, no default values are provided for representation properties as built-in definitions by any DFDL processor. This requires DFDL schemas to be explicit about the representation properties of the data they describe, and avoids any possibility of DFDL schemas that are meaningful for some DFDL processors but not others.

The properties are organized as follows:

- Common to both Content and Framing (see 11)
- Common Framing, Position, and Length (see 12)
- Simple Type Content (see 13)
- Sequence Groups (see 14)
- Choice Groups (see 15)
- Array elements and optional elements (see 16)
- Calculated Values (see 17)

Where properties are specific to a physical representation, the property name may choose to reflect this. Where properties are related to a specific logical type grouping (defined below), the property name may choose to reflect this.

A limited number of properties can take a DFDL expression which must return a value of the proper type for the property. Those properties that take an expression explicitly state in the description. Other properties do not take an expression.

The property description defines which schema component that the property may be specified on. In addition all the DFDL properties may be specified on a dfdl:format annotation.

11. **Properties Common to both Content and Framing**

Property Name	Description
byteOrder	Enum or DFDL Expression
	Valid values 'bigEndian', 'littleEndian'.
	This property can be computed by way of an expression which returns the string 'bigEndian' or 'littleEndian'. The expression must not contain forward references to elements which have not yet been processed.
	Note that there is, intentionally, no such thing as 'native' endian[16].
	This property applies to all Numbers and Calendars with representation binary. Specifically that is binary integers, all packed decimals, binary floats, binary seconds and binary milliseconds.
	This property is never used to establish the byte order for text /strings with Unicode fixed-width encodings that do not specify the byte order (UTF-16 and UTF-32). See Section 11.1 Unicode Byte Order Marks (BOM) for details.
	Annotation: dfdl:element, dfdl:simpleType, dfdl:sequence, dfdl:choice, dfdl:group
encoding	Enum or DFDL Expression
	Values are IANA charsets or CCSID[17]s, or one of a set of DFDL-specific encoding names.
	This property can be computed by way of an expression which returns an appropriate enum value. The expression must not contain forward references to elements which have not yet been processed.
	Note that there is, deliberately, no concept of 'native' encoding[18].
	Conforming DFDL v1.0 processors must accept at least 'UTF-8'', 'UTF-16', 'UTF-16BE', 'UTF-16LE', 'ASCII', and 'ISO-8859-1' as encoding names.
	Encoding names are case-insensitive, so 'utf-8' and 'UTF-8' are equivalent.
	Unicode character set encodings that do not specify a byte order (such as UTF-16 or UTF-32) can have their byte-order controlled by a document-level byte-order-mark (BOM). See Section 11.1 Unicode Byte Order Marks

---

[16] The concept of native-endian is avoided in DFDL since a DFDL schema containing such a property binding does not contain a complete description of data, but rather an incomplete one which is parameterized by characteristics of the machine and implementation where the DFDL processor is executed. In DFDL this same behavior is achieved using variables or, for example, by use of external setting of pre-defined variables to set dfdl:byteOrder.

[17] CCSID stands for Coded Character Set ID, a decimal number syntax for a coded charater set specifier. [CCSID].

[18] The concept of native character encoding is avoided in DFDL since a DFDL schema containing such a property binding does not contain a complete description of data, but rather an incomplete one which is parameterized by characteristics of the operating environment where the DFDL processor executes. In DFDL this same behavior is achieved through use of true parameterization using variables or, for example, by use of external setting of pre-defined variables to set dfdl:encoding.

	(BOM) for details.
	The encoding name 'UTF-8' is interpreted strictly and does not include variants such as CESU-8.
	DFDL-specific encoding names include:
	• US-ASCII-7-bit-packed – a US-ASCII variant where character codes occupy only 7 bits, not a full 8-bit byte.
	Implementations may allow additional DFDL-specific encoding names only for character set encodings for which there is no IANA name standard nor CCSID standard. These implementation-specific encodings must have "x-" as a prefix to their name, and they are subject to being superceded with DFDL standard names in future versions of the specification.
	Annotation: dfdl:element, dfdl:simpleType, dfdl:sequence, dfdl:choice, dfdl:group
utf16Width	Enum
	Valid values are 'fixed', variable'.
	Applies only when encoding is 'UTF-16', 'UTF-16BE', UTF16-LE' or their CCSID equivalents.
	Specifies whether the encoding 'UTF-16' should be treated as a fixed or variable width encoding. 'UTF-16' is a variable width encoding and 'UCS-2' is the fixed width subset. However it is common for users to specify 'UTF-16' when they mean when they should be specifying 'UCS-2' This property effectively converts 'UTF-16' to 'UCS-2'.
	Annotation: dfdl:element, dfdl:simpleType, dfdl:sequence, dfdl:choice, dfdl:group
ignoreCase	Enum
	Valid values are 'yes', 'no'.
	Whether mixed case data is accepted when matching delimiters and data values on input.
	This affects the behavior of matching for these properties: initiator, terminator, separator, nilValue, textStandardExponentRep, textStandardInfinityRep, textStandardNaNRep, textStandardZeroRep, textBooleanTrue, textBooleanFalse,
	On unparsing always use the delimiters or value as specified.
	Annotation: dfdl:element, dfdl:simpleType, dfdl:sequence, dfdl:choice, dfdl:group
encodingErrorPolicy	Enum
	Valid values are 'error' or 'replace'.
	This property applies whenever dfdl:encoding is applicable.
	This property provides control of how decoding and encoding errors are handled when converting the data to text, or text to data. This includes converting when scanning for delimiters, matching regular expression length or test patterns, matching textual data type representation patterns against the data, and of course isolating the text content that will become the value of an element (parsing) or constructing the content from the value (unparsing).

Comment [SMH153]: Capitalize for consistency

DONE

Comment [m154]: 2.107 Updated (OPEN)

Please review - I changed this to avoid DFDL implementations introducing synonyms. Additional names can only be added for encodings for which there is NO standard name nor CCSID.

For example "x-daffodil-edu-ncsa-illinois-ita2" might be the name of a 5 bit ITA2 encoding.

Comment [SMH155]: 2.107. DONE

Comment [SMH156]: Delete row

DONE

Comment [SMH156]: Delete row

DONE

Comment [SMH157]: Errata 2.71 says we should list the affected properties here.

DONE

Comment [m158]: Errata 3.13

(PART DONE)

Details about scanning for delimiters vs lengthKind pattern, or matching nilValues or textBooleantTrueRep are needed.
(DONE)

When parsing, an error can occur when decoding characters from their encoded form into the DFDL Infoset character set (ISO10646). This can occur due to invalid byte sequences, or not enough bytes found to make up the full encoding of a character.

If 'replace', then the Unicode replacement character (U+FFFD) is substituted for the offending errors, one replacement character for any incorrect fragment of an encoding.

If 'error' then a processing error occurs.

When unparsing, the errors that can occur when encoding characters from Unicode/ISO 10646 into the specified encoding include when no mapping is provided by the encoding specification and when there is not enough space to output the entire encoding of the character (e.g., need 2 bytes for a 2-byte character codepoint, but only 1 byte remains in the available length.)

If 'replace' then encoding-specific replacement/substitution character is output. It is a processing error if no such character is defined, and it is a processing error if there is any error when attempting to output the replacement (such as not enough room for the representation of the entire encoding of the replacement character).

If 'error' then a processing error occurs.

See Section 11.2 Character Encoding and Decoding Errors for further details.

Annotation: dfdl:element, dfdl:simpleType, dfdl:sequence, dfdl:choice, dfdl:group

Comment [SMH159]: Is this intended to be a link?

Nope. Fixed. DONE

Comment [SMH160]: Remove 'the policy is' to match other bullets

DONE

11.1   Unicode Byte Order Marks (BOM)

DFDL provides automatic detection and generation of Unicode BOMs at the document level and saves (for parsing), or retrieves (for unparsing) the BOM information from the DFDL Infoset **[unicodeByteOrderMark]** member.

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.

Comment [SMH161]: Errata 3.7: Add section to describe new Unicode BOM support

DONE

Comment [SMH162]: Wrong font

DONE

- UTF-8. If a BOM is found[19] 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.

---

[19] While UTF-8 has no true notion of byte-order, the 16-bit codepoint for a byte-order mark is often translated into a 3-byte utf-8 sequence of bytes that appear at the start of a document. This information is helpful to establish that the document is encoded in Unicode (specifically UTF-8).

- 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 must be explicitly modelled as such, otherwise if parsed as part of an xs:string it will be interpreted as a zero-width non-breaking space (ZWNBS) character.

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. If a BOM does appear at any other point, then it must be explicitly modeled as a separate element. Otherwise if parsed as part of an xs:string it will be interpreted as a ZWNBS character.

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.

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.

Comment [SMH163]: Wrong font

DONE

Comment [SMH164]: Wrong font

DONE

Comment [m165]: 3.13. *Sections 4.1.2, 11.* DONE

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

The property dfdl:encodingErrorPolicy has two possible values: 'error' and 'replace'.

11.2.1.1 dfdl:encodingErrorPolicy 'error'

If 'error', then any error when decoding characters while parsing causes a processing error. For unparsing, any error when encoding characters causes a processing 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 dfdl:lengthUnits is 'bytes', the 'not enough data' decoding 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 when decoding characters 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 dfdl:lengthUnits is 'bytes', the 'not enough data' decoding 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.)

The Unicode Replacement Character must not appear in any delimiter, pad character, nil value, regular expression, number pattern or calendar pattern, or in any other DFDL property value or attribute value where the Unicode Replacement Character would be expected in the data being parsed. It is a schema definition error if the Unicode Replacement Character appears in any of these locations of a DFDL schema, or is part of the value of an expression that returns a string to be used as the value of a DFDL property.

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'. DFDL Schema authors are advised that bounded length regular expressions can help in this case. E.g., ".{0,50}" says to match any character (including 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 decoding error. This is likely to happen for data that is (a) initially parsed by a DFDL parser with dfdl:encodingErrorPolicy 'replace', and (b) which contains some decoding errors, but (c) is nevertheless successfully parsed, (d) is written back out to a file or other data

repository, and (e) is parsed again. The written data will have replaced data errors with the Unicode Replacement Character, and so if the data is parsed again, it will no longer have errors, but will have the Unicode Replacement Character as a regular character in the data.

If dfdl:lengthUnits is '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 dfdl:lengthUnits is '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 a processing 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.)

11.2.2 Unicode UTF-16 Decoding/Encoding Non-Errors

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

- unpaired surrogate codepoint
- out-of-order surrogate codepoint pair
- surrogate codepoint pair is encountered

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

11.2.3 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 encodings have these kinds of 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.

> **Comment [SMH166]:** Is this intended to be a link?
>
> YES. Was Broken. Fixed
>
> DONE

12. Framing

Several properties are common across the various framing styles or are used to distinguish them. Generally these have to do with position and length for text, bit fields, or opaque data.

12.1 Aligned Data

Alignment properties control the leading alignment and trailing alignment regions.

When the alignment properties are applied to an array element, the properties are applied to each occurrence of the element; that is, not only to the first occurrence.

The following properties are used to define alignment rules.

Property Name	Description
alignment	Non-negative Integer or 'implicit' A non-negative number that gives the alignment required for the beginning of the item. If alignment is needed then the size of the **AlignmentFill** grammar region will be non-zero if the item must be aligned to a boundary. 'implicit' specifies that the natural alignment for the representation type is used. See the table of implicit alignments Table 14 Implicit Alignment in bits for simple elements. The 'implicit' alignment of complex elements and groups is the alignment of its child with the greatest alignment. If alignment is 'implicit' then alignmentUnits is ignored. For textual data, minimum alignment is mandated by the character-set encoding, and this property must be 'implicit' or set to a multiple of the character-set's mandatory alignment. See Section 12.1.2. Annotation: dfdl:element, dfdl:simpleType, dfdl:sequence, dfdl:choice, dfdl:group
alignmentUnits	Enum Valid values are 'bits' or 'bytes' Scales the alignment so alignment can be specified in either units of bits or units of bytes. Only used when dfdl:alignment not 'implicit' Annotation: dfdl:element, dfdl:simpleType, dfdl:sequence, dfdl:choice, dfdl:group
fillByte	DFDL String Literal A single byte specified as a DFDL byte value entity or a single character. If a character is specified, it must be a single-byte character in the applicable encoding. Used on unparsing to fill empty space such as between two aligned elements. Used to fill these regions specified in the grammar: **RightPadOrFill**, **ElementUnused**, **ChoiceUnused**, **LeadingSkip**, **AlignmentFill**, and **TrailingSkip**. Annotation: dfdl:element, dfdl:simpleType, dfdl:sequence, dfdl:choice, dfdl:group
leadingSkip	Non-negative Integer

	A non-negative number of bytes or bits, depending on dfdl:alignmentUnits, to skip before alignment is applied. Gives the size of the grammar region having the same name.
	Annotation: dfdl:element, dfdl:simpleType, dfdl:sequence, dfdl:choice, dfdl:group
trailingSkip	Non-negative Integer
	A non-negative number of bytes or bits, depending on dfdl:alignmentUnits, to skip after the element, but before considering the alignment of the next element. Gives the size of the grammar region having the same name.
	If dfdl:trailingSkip is specified when dfdl:lengthKind is 'delimited' then a dfdl:terminator must be specified.
	Annotation: dfdl:element, dfdl:simpleType, dfdl:sequence, dfdl:choice, dfdl:group

Comment [SMH173]: I believe that an in-progress errata disallows endOfParent with trailingSkip.

Comment [m174]: (DONE)

There are two properties which control the data alignment by controlling the length of the **AlignmentFill** region

- alignment - an integer 1 or greater
- alignmentUnits - bits or bytes

An element's representation is aligned to N units if P is the first position in the representation and P mod N = 1. When parsing, the position of the first unit of the data stream is 1.

For example, if alignment=4, and alignmentUnits='bytes', then the element's representation must begin at 1 or 1 plus a multiple of 4 bytes. I.e., 1, 5, 9, 13, 17 and so on.

The length of the **AlignmentFill** region is measured in bits. If alignmentUnits is 'bytes' then we multiply the alignment value by 8 to get the *bit alignment*, B. If the current position (first position after the end of the previous element) is bit position N, then the length of the **AlignmentFill** region is the smallest non-negative integer L such that (L + N) mod B = 1. The position of the first bit of the aligned element is P = L + N.

To avoid ambiguity when parsing, optional elements and variable array elements where the minimum number of occurrences is zero cannot have alignment properties different from the items that follow them. It is a schema definition error otherwise. This avoids the possibility that the following item is incorrectly parsed as if it were a valid optional element occurrence or a valid variable array element occurence.

The **LeadingSkip** and **TrailingSkip** regions length are controlled by two properties of corresponding names and the dfdl:alignmentUnits property.

12.1.1 Implicit Alignment

When dfdl:alignment is 'implicit' the following alignment values are applied for each logical type.

Type	Alignment	
	text	**binary**
String		Not applicable
Float		32
Double		64

Decimal, Integer, nonNegativeInteger	Encoding Specific (usually 8 bits, with exceptions: See Section 12.1.2)	Packed decimals: 8	binary: 8
Long, UnsignedLong			binary: 64
Int, UnsignedInt			binary: 32
Short, UnsignedShort			binary: 16
Byte, UnsignedByte			binary: 8
DateTime			binarySeconds: 32, binaryMilliseconds:64
Date			binarySeconds: 32, binaryMilliseconds:64
Time			binarySeconds: 32, binaryMilliseconds:64
Boolean		32	
HexBinary	Not applicable	8	

Table 14 Implicit Alignment in bits

Note: The above table specifies the implicit alignment in bits, but this does not imply that dfdl:lengthUnits 'bits' can be specified for all simple types. Rather, dfdl:alignmentUnits and dfdl:lengthUnits are independent and have their own rules for when they are applicable.

12.1.2   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 encoding 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-specified 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 required character set encodings in DFDL have 8-bit/1-byte alignment.

Some implementations may include additional encodings which have other alignments. The DFDL standard specifies a name for one such character set encoding though conforming implementations are not required to support this encoding:

• 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).

Implmentations may also provide identifiers for non-standard encodings, and these will have their own specific alignments as well.

Note the 16-bit and 32-bit Unicode character set encodings UTF-16, UTF-16BE, UTF-16LE, UTF-32, UTF-32BE, UTF-32LE, all have 8-bit/1-byte alignment.

Comment [SMH175]: Errata 2.32: Clarify DONE

Comment [m176]: 2.107 Updated (OPEN) Modified language here since description of encoding above now allows for implementation-specific encodings (names begin with "x-") for example "x-daffodil-ita2" might be the name of a 5 bit ITA2 encoding.

Comment [m177]: Errata 2.95 Errata 2.107 DONE

12.2   Properties for Specifying Delimiters

The following properties apply to all objects that use text delimiters to delimit, that is, to initiate and/or terminate data. Delimiters can apply to binary data; however they are most often called 'text' delimiters because the concept is much more commonly used for textual data formats.

Property Name	Description
initiator	List of DFDL String Literals or DFDL Expression
	Specifies a whitespace separated list of alternative literal strings one of which marks the beginning of the element or group of elements. This property can be computed by way of an expression which returns a string containing a whitespace separated list of DFDL String Literals. The expression must not contain forward references to elements which have not yet been processed.
	The *Initiator* region contains one of the initiator strings defined by dfdl:initiator.
	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).
	When an initiator is specified, it is a processing error if the component is required and one of the values is not found.
	If dfdl:initiator is "" (the empty string), then the *Initiator* region is of length zero, and no initiator is expected. It is not permitted for an expression to return an empty string. That is a schema definition error.
	On unparsing the first initiator in the list is automatically inserted into the *Initiator* region.
	If dfdl:ignoreCase is 'yes' then the case of the string is ignored by the parser.
	Annotation: dfdl:element, dfdl:simpleType, dfdl:sequence, dfdl:choice, dfdl:group

Comment [SMH178]: Just say 'all objects' and remove the sequence/choice stuff

DONE

Comment [m179]: I revised the wording of this somewhat to reflect that the longest-match behavior is not a static sort of the string literal tokens, but it is the longest match against the actual data.

Comment [SMH180]: Errata 2.63. **DONE**

Comment [SMH181]: Errata 2.70 specifies the parsing matching algorithm

DONE

| terminator | List of DFDL String Literals or DFDL Expression
Specifies a whitespace separated list of alternative text strings that one of which marks the end of an element or group of elements. The strings MUST be searched for in the longest first order.
This property can be computed by way of an expression which returns a string of whitespace separated list of values. The expression must not contain forward references to elements which have not yet been processed.
The **Terminator** region contains the terminator string.

If dfdl:terminator is "" (the empty string), then the terminator region is of length zero, and no terminator is expected. It is not permitted for an expression to return an empty string, that is a schema definition error.

When parsing, the list of values is processed in a greedy manner, meaning it takes all the terminators, 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 terminator with the longest match is the one that is selected as having been 'found', with length-ties being resolved so that the matching terminator is selected that is first in the order written in the schema. Once a matching terminator is found, no other matches will be subsequently attempted (ie, there is no backtracking).

When a terminator is expected it is a processing error if no matching terminator is found. However, if dfdl:documentFinalTerminatorCanBeMissing is specified then it is not an error if the last terminator in the data stream is not found.

On unparsing the first terminator in the list is automatically inserted in the **Terminator** region.
If dfdl:ignoreCase is 'yes' then the case of the string is ignored by the parser.
Annotation: dfdl:element, dfdl:simpleType, dfdl:sequence, dfdl:choice, dfdl:group |

Comment [SMH182]: Errata 2.63. **DONE**

Comment [m183]: I revised the wording of this somewhat to reflect that the longest-match behavior is not a static sort of the string literal tokens, but it is the longest match against the actual data.

Comment [SMH184]: Insert paragraph break

DONE

Comment [SMH185]: Errata 2.70 specifies the parsing matching algorithm

DONE

emptyValueDelimiterPolicy	Enum
	Valid values are 'none', 'initiator', 'terminator' or 'both'
	Indicates that when an element in the data stream is empty, an initiator (if one is defined), a terminator (if one is defined), both an initiator and a terminator (if defined) or neither must be present.
	Ignored if both dfdl:initiator and dfdl:terminator are "" (empty string).
	'initiator' indicates that, on parsing, if the content region (which can be either the SimpleContent region or the ComplexContent region defined in Section 9.2) is empty then the dfdl:initiator must be present. It also indicates that on unparsing when the content region is empty that the dfdl:initiator will be output.
	'terminator' indicates that, on parsing, if the content region is empty then the dfdl:terminator must be present. It also indicates that on unparsing when the content region is empty the dfdl:terminator will be output.
	'both' indicates that, on parsing, if the content region is empty both the dfdl:initiator and dfdl:terminator must be present. On unparsing when the content region is empty the dfdl:initiator followed by the dfdl:terminator will be output.
	'none' indicates that if the content region is empty neither the dfdl:initiator or dfdl:terminator must be present. On unparsing when the content region is empty nothing will be output.
	It is a schema definition error if emptyValueDelimiterPolicy set to 'none' or 'terminator' when the parent xs:sequence has dfdl:initiatedContent 'yes'.
	This property plays an important role in establishing empty representation. See 9.2.2 Empty Representation for details.
	Annotation: dfdl:element, dfdl:simpleType
documentFinalTerminator CanBeMissing	Enum
	Valid values are 'yes', 'no'
	When the documentFinalTerminatorCanBeMissing property is true, then when an element is the last element in the data stream, then on parsing, it is not an error if the terminator is not found.
	For example, if the data are in a file, and the format specifies lines terminated by the newline character (typically LF or CRLF), then if the last line is missing its newline, then this would normally be an error, but if documentFinalTerminatorCanBeMissing is true, then this is not a processing error.
	On unparsing the terminator is always written out regardless of the state of this property.
	Annotation: dfdl:format (but applies to elements only)

Comment [m186]: MJB Errata 2.6 DONE

Comment [SMH187]: DONE: Needs reviewing as part of action 140

Comment [m188]: (DONE) language as is OK.

outputNewLine	DFDL String Literal or DFDL Expression
	Specifies the character or characters that will be used to replace the %NL; character class entity during unparse
	It is a schema definition error if any of the characters are not in the set of characters allowed by the DFDL entity %NL; Only individual characters or the %CR;%LF; combination are allowed.
	It is a schema definition error if the DFDL entity %NL; is specified
	This property can be computed by way of an expression which returns a DFDL string literal. The expression must not contain forward references to elements which have not yet been processed.
	Annotation: dfdl:element, dfdl:simpleType, dfdl:sequence, dfdl:choice, dfdl:group

> **Comment [SMH189]:** 2.94. DONE

12.3   Properties for Specifying Lengths

These properties are used to determine the representation length of an element and apply to elements of all types (simple and complex).

Property Name	Description
lengthKind	Enum
	Controls how the representation length of the component is determined.
	Valid values are: 'explicit', 'delimited', 'prefixed', 'implicit', 'pattern', 'endOfParent'
	A full description of each enumeration is given in the later sections.
	'explicit' means the length of the element is given by the dfdl:length property
	'delimited' means the element length is determined by scanning for a terminator or separator.
	'prefixed' means the length of the element is given by an immediately preceding PrefixLength data region the format of which is specified using prefixLengthType.
	'implicit means the length is to be determined in terms of the type of the element and its schema-specified properties if any.
	'pattern' means the length of the element is given by scanning for a regular expression specified using the dfdl:lengthPattern property.
	'endOfParent' means that the length extends to the end of the containing (parent) construct.
	Annotation: dfdl:element, dfdl:simpleType
lengthUnits	Enum
	Valid values 'bytes', 'characters', ''bits'.
	Specifies the units to be used whenever a length is being used to extract or write data. Applicable when lengthKind is 'explicit', 'implicit' (for xs:string and xs:hexBinary) or 'prefixed'.
	'characters' may only be used for simple elements with representation 'text' and complex elements where all the simple child elements must be

> **Comment [SMH190]:** Errata 2.33: Improve the descriptions for delimited, implicit and endOfParent
>
> DONE

> **Comment [SMH191]:** Wording should reflect both parsing and unparsing.
>
> Handled in later Section.
> DONE

> **Comment [SMH192]:** I think 'item' should be 'element' (4 occurrences)
>
> DONE

> **Comment [SMH193]:** Needs improving as could be read that the preceding field is an element in its own right.
>
> DONE

> **Comment [SMH194]:** Font, and 'using the dfdl:prefixLengthType property' for consistency

dfdl:representation 'text', dfdl:lengthUnits 'characters' and the same dfdl:encoding as the parent.

'bits' may only be used for xs:boolean, xs:byte, xs:short, xs:int, xs:long, xs:unsignedByte, xs:unsignedShort, xs:unsignedInt, and xs:unsignedLong simple types with representation 'binary'.

Comment [m195]: Errata 2.90 Done

For signed types, the physical bits are interpreted as a twos-complement integer.

 Annotation: dfdl:element, dfdl:simpleType

12.3.1 dfdl:lengthKind 'explicit'

Comment [m196]: *Errata 2.100 - Section 12.3.1.* DONE

When lengthKind='explicit' the length of the item is given by the dfdl:length property, and is measured in units given by dfdl:lengthUnits. Used on parsing and unparsing.

When unparsing an element with dfdl:lengthKind 'explicit' and where dfdl: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 dfdl:lengthKind 'prefixed'. See Section 12.3.4.

When parsing, data is extracted without regard to any in-scope delimiters.

Property Name	Description
length	Non-negative Integer or DFDL Expression.
	Only used when lengthKind is 'explicit'.
	Specifies the length of this element in units specified by dfdl:lengthUnits.
	This property can be computed by way of an expression which returns a non-negative integer. The expression must not contain forward references to elements which have not yet been processed.
	Annotation: dfdl:element, dfdl:simpleType

When dfdl:lengthKind 'explicit', the method of extracting data is described in section: 12.3.7 Elements of Specified Length

12.3.2 dfdl:lengthKind 'delimited'

On parsing, the length of an element with dfdl:lengthKind 'delimited' is determined by scanning the datastream for the delimiter.

The data stream is scanned for any of

- the element's terminator (if specified)
- an enclosing construct's separator or terminator
- the end of an enclosing element designated by its known length
- the end of the data stream

dfdl:lengthKind 'delimited' may be specified for

- elements of simple type with dfdl:representation 'text'
- elements of number or calendar simple type with dfdl:representation 'binary' that have a packed decimal representation
- elements of type xs:hexBinary
- elements of complex type.

The rules for resolving ambiguity between delimiters are:

1. When two delimiters have a common prefix, the longest delimiter is tried first.
2. When two delimiters have exactly the same length, but on different schema components, the innermost (most deeply nested) delimiter is tried first.
3. When the separator and terminator on a group have the same value, then at a point in the data where either the separator or terminator could be found, the separator is tried first. (Speculative execution may try the terminator subsequently).
4. If the length of the delimiters cannot be determined because character class entities (which are variable length) are being used then the delimiters must each be matched against the data, and the longest matching delimiter is taken as the match for the delimiter.
5. Ties (same matched length) are broken by giving a separator priority over a terminator of a sequence, or by choosing the innermost, or first in schema order.

When unparsing a simple element with text representation, the length in the data stream is the length of the content region, padded to dfdl:textOutputMinLength or xs:minLength if dfdl:textPadKind is 'padChar'.

When unparsing a simple element with binary representation, then for hexBinary the length is the number of bytes in the infoset value padded to xs:minLength using dfdl:fillByte, and for the other types the length is the minimum number of bytes to represent the value and any sign.

When unparsing a complex element, the length is that of the ComplexContent region.

12.3.2.1 Simple Elements of Specified Length within Delimited Constructs

When a simple or complex element has a specified length or lengthKind 'pattern' then delimiter scanning is suspended for the duration of the processing of that element.

This allows formats to be parsed which are delimited, but have nested elements which contain non-character data.

12.3.2.2 Delimited Binary Data

Formats involving binary data, most notably packed decimals, can use delimiters but care must be taken that the delimiters cannot match data represented in these formats. In particular, the

Comment [m197]: Errata 2.97. *Done*

Comment [SMH198]: Errata 2.34: Clarify **DONE**

Comment [m199]: Changed to new terminology.

Comment [m200]: Errata 2.82 (part thereof)

Comment [m201]: DONE
This contradicts the table in the section on binaryNumberRep which specifically allows binaryNumberRep='binary' for decimal, integer, and nonNegativeInteger types.

Withdrawn (DONE)

SMH: Not sure why this contradicts? These rules are now expressed in Table 16 so might be better to ref that instead. If not, this para needs augmenting with stuff about binaryCalendarRep which is missing. (DONE)

Comment [SMH202]: 2.97 Tidied and corrected. DONE

Comment [SMH203]: The table contains rows that are not possible. There are restrictions on what binary data is allowed to have lengthKind 'delimited'. However, later sections for 'pattern' and 'prefixed' refer to this information. Need to think how best to organize the table.

Table moved down to section on prefixed lengthKind. Comment did not go with… drat. (DONE)

Comment [m204]: Errata 2.82 (part thereof) DONE

Comment [SMH205]: Changed lengths to bits to match all other tables
(Note: table this was a comment on has been moved.)
DONE

Comment [m206]: DONE: specified length means fixed, explicit, prefixed. But why can't one have a contained element of type pattern? When searching for the pattern match, delimiter scanning is suspended (already says this in section on lengthKind pattern), so this will naturally nest just like specified lengths.
SMH: Agree. Tim suggests that the sections on explicit, etc, should also say that data is extracted without regard to any in-scope delimiters.

Fixed (DONE)

Comment [SMH207]: Errata 3.3 specifies the rules for this

DONE

Comment [m208]: I think this is DONE. Is there more to say here. Not much in Errata 3.3 to put here.

(DONE) until reviewed.

delimiters must be chosen with knowledge that BCD data can contain any byte both of whose nibbles are 0 to 9 (that is, excluding A to F). Packed data adds bytes with a sign indicator, that is, a nibble in the range A to F.

General binary data can contain any bit pattern whatsoever, so delimiters for numbers and calendars with dfdl:representation 'binary' are disallowed, with the specific exception of packed decimals. Delimiters and lengthKind 'delimited' are also allowed for type xs:hexBinary.

Carefully chosen delimiters can avoid the byte values that can appear within the data.

12.3.3 dfdl:lengthKind 'implicit'

When dfdl:lengthKind is 'implicit', the length is determined in terms of the type of the element and its schema-specified properties.

For complex elements, 'implicit' means the length is determined by the combined lengths of the contained children, that is the ComplexContent region.

For simple elements the length is fixed and is given in Table 15 Length in Bits for SimpleTypes when dfdl:lengthKind='implicit' .

Type	Length		
	text	**binary**	
String	maxlength gives length in either characters or bytes depending on the value of dfdl:lengthUnits.	Not applicable	
Float	Not allowed	32	
Double	Not allowed	64	
Decimal, Integer, nonNegativeInteger	Not allowed	packed decimal: Not allowed	binary: Not allowed
Long, UnsignedLong	Not allowed		binary: 64
Int, UnsignedInt	Not allowed		binary: 32
Short, UnsignedShort	Not allowed		binary: 16
Byte, UnsignedByte	Not allowed		binary: 8
DateTime	Not allowed		binarySeconds: 32, binaryMilliseconds:64
Date	Not allowed		binarySeconds: 32, binaryMilliseconds:64
Time	Not allowed		binarySeconds: 32, binaryMilliseconds:64
Boolean	Length of longest of textBooleanTrue and	32	

Comment [m209]: DONE
Do we disallow binaryNumberRep='binary' with delimited lengthKind?
SMH: Agreed to disallow 'binary'.
MikeB: revised misleading text.
Inconsistent with scannability requirement, stipulated above.

Comment [SMH210]: Errata 2.9 clarifies what this mean for lengthUnits 'bytes'

DONE

Comment [SMH211]: Might be better as part of the final bullet in the list below the table?

DONE there also.

Comment [Steve Han212]: Errata 2.9 clarifies what this mean for lengthUnits 'bytes'

DONE

Comment [Mike Beck213]: *Reply to Steve Hanson (03/01/2012, 11:41): "..."*
DONE

	textBooleanFalse values	
HexBinary	Not applicable	maxlength

Table 15 Length in Bits for SimpleTypes when dfdl:lengthKind='implicit'

- 'Not Allowed' means that there is no implicit length for the combination of simple type and representation and it is a schema definition error if dfdl:lengthKind='implicit' is specified.

- packed decimal means binaryNumberRep is 'packed', 'bcd', or 'ibm4690Packed'

- binary means binaryNumberRep is 'binary'

- binarySeconds means binaryCalendarRep is 'binarySeconds'

- binaryMilliseconds means binaryCalendarRep is 'binaryMilliseconds'

- maxLength means xs:maxlength, and gives length for strings in either characters or bytes depending on dfdl:lengthUnits.

Note: Specifying the implicit length in bits does not imply that dfdl:lengthUnits 'bits' can be specified for all simple types.

When dfdl:lengthKind is 'implicit', the method of extracting data that is described in section: 12.3.7 Elements of Specified Length

12.3.4   dfdl:lengthKind 'prefixed'

When dfdl:lengthKind is 'prefixed' the length of the element is given by the PrefixLength region specified using prefixLengthType. The property prefixIncludesPrefixLength also can be used to adjust the length appropriately.

When dfdl:lengthKind is 'prefixed' the method of extracting data that is described in section: 12.3.7 Elements of Specified Length

Data is extracted without regard to any in-scope delimiters.

Property Name	Description
prefixIncludesPrefixLength	Enum
	Valid values are 'yes', 'no'
	Whether the length given by a prefix includes the length of the prefix as well as the length of the content region (which can be either the SimpleContent region or the ComplexContent region defined in Section **Error! Reference source not found.**).)
	Used only when dfdl:lengthKind 'prefixed'.
	Annotation: dfdl:element, dfdl:simpleType
prefixLengthType	QName
	Name of a simple type derived from xs:integer or any subtype of it.
	This type, with its DFDL annotations specifies the representation of the length prefix, which is in the PrefixLength region.
	It is a schema definition error if the xs:simpleType specifies dfdl:lengthKind 'delimited' or 'endOfParent' or 'pattern' or 'explicit'

Comment [m214]: Errata 2.6 DONE

Comment [SMH215]: 'prefixed'

DONE

	where length is an expression, a dfdl:outputValueCalc, a value for dfdl:initiator or dfdl:terminator other than empty string, dfdl:alignment other than '1', or dfdl:leadingSkip or dfdl:trailingSkip other than '0'.[20] Annotation: dfdl:element, dfdl:simpleType

The representation of the element is in two parts.

1. The 'prefix length' is an integer which specifies the length of the element's content. The representation of the length prefix is described by a simple type which is identified using the dfdl:prefixLengthType property.

2. The content of the element.

When parsing, the length of the element's content is obtained by parsing the simple type specified by dfdl:prefixLengthType to obtain an integer value. Note that all required properties must be present on the specified simple type or defaulted because there is no element declaration to supply any missing required properties.

If the dfdl:prefixIncludesPrefixLength property is 'yes' then the length of the element's content is the value of the prefix length minus the length of the content of the prefix length.

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.

The DFDL properties that specify the format of the prefix come from annotations directly on the prefixLengthType's type definition, and from the default format annotation for the schema document containing the definition of that type. . 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.

When unparsing, the length of the element's content region must be determined first as described below. Then the value of the prefix length must be adjusted using dfdl:prefixIncludesPrefixLength.

Then the prefix length can be written to the data stream using the properties on the dfdl:prefixLengthType, and finally the element's content can be written to the data stream.

Consider this example:

```
<xs:element name="myString" type="xs:string"
                   dfdl:lengthKind="prefixed"
                   dfdl:prefixIncludesPrefixLength="false"
                   dfdl:prefixLengthType="packed3"/>

<xs:simpleType name="packed3"
           dfdl:representation="binary"
           dfdl:binaryNumberRep="packed"
           dfdl:lengthKind="explicit"
           dfdl:length="2" >
```

---

[20] Note: These restrictions are in place because it would not be sensible for these properties to reside on a type used for a prefixLengthType. These properties are irrelevant to the format of a SimpleContent region (they control regions that are part of other surrounding regions present only for a full element). However, the dfdl:simpleType annotation allows these properties to be present on a simpleType definition. It is only in this specific usage of simpleType to define a type for use as a prefixLengthType that these properties are disallowed. Normal usage of a simpleType from an element would combine these with properties on the element to create a combined set, and in that case, these properties might be expressed on the simpleType, but be used when the combined set of properties is created.

Comment [m216]: Errata 2.14 as revised DONE

Comment [SMH217]: Errata 2.14 adds more restrictions

DONE

Comment [m218]: DONE

Review. Is this proper use of the term representation, ie., the prefix will include the length of any framing of the prefix?

SMH: Correct term is 'content' according to latest grammar.

Comment [m219]: Sentence about lengthKind pattern deleted

Revised errata 2.14 and 2.76
DONE.

Comment [m220]: Errata 2.76
(DONE)

Comment [m221]: Errata 2.91 as revised. Wording improvement.
DONE

Comment [SMH222]: Errata 2.14. Stops recursive prefix length types.

Comment [m223]: Done – see section below.

```
   <xs:restriction base="integer" />
</xs:simpleType>
```

In the above, the string has a prefix length of type 'packed3' containing 3 packed decimal digits.

The property dfdl:prefixIncludesPrefixLength is an enumeration which allows the length computation to be varied to include or exclude the length of the prefix element itself.

The prefix length's value contains the length measured in units given by dfdl:lengthUnits.

When parsing, if the lengthUnits are bits, then any number of bits can be in the representation.However, the same is not true when unparsing. The DFDL Infoset does not store the number of bits in a number, so the number of bits will always be a multiple of 8 bits.

> **Comment [m224]:** Errata 2.78 (DONE)

When unparsing, the value of the prefix is computed automatically by obtaining the length of the element's content.

For a simple element with text representation, the length is computed as for lengthKind 'delimited'.

For a simple element with binary representation, the length is given in the table below.

For a complex element, the length is that of the ComplexContent region.

> **Comment [SMH225]:** You need to restate the rule for text from delimited, and add a table for binary (the table is not the same as for delimited as there are more allowed types)

Type	Length	
String	Not applicable	
Float	32	
Double	64	
Decimal, Integer, NonNegativeInteger	Minimum number of bytes to represent significant digits and sign	
Long, UnsignedLong	packed decimal: as Decimal	binary: 64
Int, UnsignedInt		binary: 32
Short, UnsignedShort		binary: 16
Byte, UnsignedByte		binary: 8
DateTime		binarySeconds: 32, binaryMilliseconds:64
Date		binarySeconds: 32, binaryMilliseconds:64
Time		binarySeconds: 32, binaryMilliseconds:64
Boolean	32	
HexBinary	Number of bytes in the infoset value padded to xs:minLength using dfdl:fillByte if necessary.	

> **Comment [m226]:** DONE
> Should we disallow binaryNumberRep='binary' for these types? If so, we need to say so here.
>
> Agreed not to limit this. DONE

> **Comment [SMH227]:** I think this should say "Minimum number of bytes to represent pattern digits and sign". Same for Date and Time.

Table 7 Unparse Lengths (in Bits) for dfdl:lengthKind 'prefixed'

> **Comment [SMH228]:** Table is 16

> **Comment [SMH229]:** Changed lengths to bits to match all other tables (DONE)

12.3.4.1 Nested Prefix Lengths[21]

It is possible for a prefix length, as specified by prefixLengthType, to itself have a prefix length

It is a schema definition error if this nesting exceeds 1 deep. That is, an element can have a prefix length, which defines a PrefixLength region (see Section **Error! Reference source not found.**). The PrefixLength region can itself have a type which also specifies a prefix length, thereby defining a PrefixPrefixLength region. It is a schema definition error unless the type associated with the PrefixPrefixLength is different from the type associated with the PrefixLength.

12.3.5 dfdl:lengthKind 'pattern'

The dfdl:lengthKind 'pattern' means the length of the element is given by a regular expression specified using the dfdl:lengthPattern property. The DFDL processor scans the data stream to determine a string value that is the longest match to a regular expression. The pattern is only used on parsing,

When dfdl:lengthKind is 'pattern', it means that delimiter scanning is turned off and in-scope delimiters are not looked for within or between elements.

Property Name	Description
lengthPattern	DFDL Regular Expression.
	Only used when lengthKind is 'pattern'.
	Specifies a regular expression that, on parsing, is executed against the datastream to determine the length of the element.
	The data stream beginning at the starting offset of the content region (which can be either the SimpleContent region or the ComplexContent region defined in Section **Error! Reference source not found.**)) of the element is interpreted as a stream of characters in the encoding of the element, and the regular expression contained in the dfdl:lengthPattern property is executed against that stream of characters. When the element is complex this is the dfdl:encoding of the complex element itself. Child content and framing contained within the element must be scannable (see below).
	Escape schemes are not applied when executing the regular expression.
	It is a processing error if conversion of data into a string based on the character set encoding causes an error due to illegal bit patterns that are not legal for the encoding. See also encodingErrorPolicy in Section 11 Properties Common to both Content and Framing.
	Annotation: dfdl:element, dfdl:simpleType

On unparsing the behavior is the same as for lengthKind 'prefixed'.

12.3.5.1 Pattern-Based Lengths - Scanability

Any element (complex or simple type) may have a dfdl:lengthKind of 'pattern' as long as the data in the content region (which can be either the SimpleContent region or the ComplexContent

---

[21]   This feature allows DFDL to describe the needed "one more level" of prefix that is needed for modeling an ASN.1 format, but without the complexities of general recursion

Comment [m230]: Errata 2.14 Done

Comment [SMH231]: To be consistent use "It is possible for a prefix length, as specified by prefixLengthType, to itself have a prefix length"

DONE

Comment [m232]: Errata 3.3 DONE

Comment [m233]: Errata 2.6 Done

Comment [SMH234]: After action 140 we will likely remove these words as they will be common to all zero-length elements.

SMH: DONE. – now covered in 9.2

Comment [m235]: (OPEN)

Comment [m236]: Changed to prefixed which is where length table now resides because delimited was simplified.

But prefixed just refers to 'delimited', because this is only for text representation (per section immediately following 12.3.5.1)

However, I am not sure this is correct, as lengthKind Pattern does not have any sort of stored length, nor any sort of delimiter. It depends only on the examination of the data.

(DONE)
Referring to delimited is correct. The rules are the same – ouput the data padded to minimum length is specified.

Comment [m237]: Errata 2.82 Done.

Comment [SMH238]: Table can be deleted as per errata 3.9.

Comment [SMH239]: This table affected by an in-progress errata.

Comment [m240]: (DONE) Note Table removed. Refers to the earlier section on lengthKind prefixed.

Comment [SMH241]: The rules are different for delimited and pattern, as delimited allows packed and BCD. Should pattern also allow packed and BCD?

DONE – split into 2 sections. Moved the delimited part up into the main section on delimited length kind above as a section on delimited binary data.

Note: the answer is 'no' we don't allow pattern to run over any binary representations, even bcd or packed.

Comment [SMH242]: Does this also apply to endOfParent?

Comment [m243]: I don't think so.

Comment [Mike Beck244]: Errata 2.6 DONE

region defined in Section **Error! Reference source not found.**) of the element is legal in the stated encoding of that element. Data which satisfies this is referred to as *scannable* data.

Specifically, data is scannable and length kind 'pattern' can be used only for:

o   elements of simple type with representation 'text'

o   elements of complex type where:

1. all simple child elements must have representation 'text' and have the same encoding as the parent complex element, and

2. all complex child elements must themselves follow 1 and 2 (recursively).

12.3.6 dfdl:lengthKind 'endOfParent'

The dfdl:lengthKind 'endOfParent' means that the element is terminated by the end of the data stream or the end of an enclosing complex element, sequence or choice. The enclosing component must have an identifiable end point, such as a known length, a delimiter, a pattern, or end of data stream. The enclosing component does not have to be the immediate parent of the element, but there must be no other elements defined between the element specifying 'endOfParent' and the end of the enclosing component.

A dfdl:lengthKind of 'endOfParent' can only be used on simple and complex elements in the following locations:

• When the immediate parent is a sequence, on the final element in the sequence

• When the immediate parent is a choice, on any element that is a branch of the choice

• A simple type or global element declaration referenced by one of the above.

• A global element declaration that is the document root.

It is a schema definition error if:

• the element has a terminator.

• the element has dfdl:trailingSkip not equal to 0.

• the element has maxOccurs > 1.

• any other element is defined between this element and the end of the enclosing component.

• parent element lengthKind is 'implicit' or 'delimited'.

If the element is in a sequence then it is a schema definition error if:
• the sequence is not the content model of a complex type definition
• the separatorPosition of the sequence is 'postFix'
• the sequenceKind of the sequence is not 'ordered'
• the sequence has a terminator
• there are floating elements in the sequence
• the sequence has a non-zero trailingSkip

If the element is in a choice where choiceLengthKind is 'implicit' then it is a schema definition error if:
• the choice is not the content model of a complex type definition
• the choice has a terminator
• the choice has a non-zero trailingSkip

A simple element must have either type xs:string, or dfdl:representation 'text', or type xs:hexBinary, or dfdl:representation 'binary' and a packed decimal representation.

The dfdl:lengthKind 'endOfParent' is used when the length of an element is defined by an enclosing element . For example, the parent is a fixed length element then an element with dfdl:lengthKind 'endOfParent' will consume all the remaining data up to the length of the parent. A dfdl:lengthKind 'endOfParent' can also be used to allow the last element to consume the data up to the end of the data stream.

This is distinct from situation where the lengths of the elements of a sequence are known but are not sufficient to fill the fixed length parent. In that case the remaining data are ignored on parsing and filled with dfdl:fillByte on unparsing.

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

A complex element can have 'endOfParent'. If so then its last child element can be any dfdl: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:

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

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

Comment [m253]: Errata 2.72 DONE

Comment [m254]: Errata 2.73 DONE

Comment [m255]: Errata 2.72 as revised DONE.

12.3.7   Elements of Specified Length

An element has a specified length when dfdl:lengthKind is 'explicit', 'implicit' (simple type only) or 'prefixed'. The units that the length represents are specified by the dfdl:lengthUnits property, except when dfdl:lengthKind is 'implicit' and the simpleType is not xs:string or xs:hexBinary.

If dfdl:lengthUnits='bytes' then the value of the length property gives the exact length in bytes.

If dfdl:lengthUnits='characters' then the length in bytes will depend on the encoding of the characters. If the encoding property specifies a fixed-width encoding then the length in bytes is (character width*length). If the encoding property specifies a variable-width encoding then the length in bytes will depend on the actual characters in the element's value. The dfdl:lengthUnits 'characters' may only be used when dfdl:representation is 'text', it is a schema definition error otherwise.

If dfdl:lengthUnits='bits' then the value of the length property gives the exact length in bits. 'Bits' may only be used for a limited set of schema types. See section: 12.3.7.2 Length of Bit Fields

Using specified length, it is possible for an element to have content length longer than needed to represent just the data value. For example, a simple text element may be padded in the **RightPadOrFill** region if the data is not long enough.

When an element has specified length, delimiter scanning is turned off and in-scope delimiters are not looked for within or between elements.

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 dfdl:fillByte property.

The dfdl:fillByte is *not* used to trim an element of type xs:hexBinary when parsing.

An element of specified length with dfdl:lengthKind 'implicit' or 'explicit' where dfdl:length is not an expression has a known length when unparsing. However, an element of specified length with dfdl:lengthKind 'prefixed' or 'explicit' where dfdl:length is an expression is considered to have a *variable* length when unparsing.Specifically:

Comment [SMH256]: Simple type only.

Wording changed.

DONE

Comment [SMH257]: I think this should be removed as we are just talking specified lengths here.

DONE

Comment [m258]: Errata 3.3 DONE

Comment [m259]: Errata 2.87 Done

- For dfdl:lenghKind 'explicit' (expression), the processor cannot automatically determine in what way the length information is to be stored. Normally the values of one or more elements would be computed using dfdl:outputValueCalc, with expressions that measure the length of the element using functions such as dfdl:contentLength or dfdl:valueLength.

- For dfdl:lengthKind 'prefixed' the processor automatically determines the value to store in the prefix, based on the length of the infoset element, and the properties which modify the interpretation of the prefix length value, such as dfdl:prefixIncludesPrefixLength.

12.3.7.1 Length of Simple Elements with dfdl:representation 'text'

Textual data is defined to mean either data of type string or data where the representation property is 'text'.

12.3.7.1.1 Character Width

The *width* of a character is the length of its representation in bytes and depends on the dfdl:encoding property.

Character encodings are themselves either intrinsically fixed or variable width, but this is modified by additional properties.

fixed, 1 byte (e.g., ASCII)	fixed, 2 byte (e.g., UCS-2)	fixed, 4 byte (e.g., UTF-32)	variable (e.g., Shift_JIS, UTF-8, UTF-16)
1	2	4	Variable width. Min = 1, Max = encoding dependent

Table 16 Character Widths

We define the term *fixed width encoding* to mean an encoding and associated other representation properties where the value in the bottom row of the above table is a fixed integer.

The term *variable width encoding* is the opposite. In a variable width encoding, the characters have a minimum and a maximum length. The maximum depends on the encoding, but is typically either 3 (Shift-JIS), or 4 (UTF-8).

UTF-16, UTF16LE and UTF16BE are variable width encodings, however when dfdl:utf16Width is 'fixed' they are treated as a 2 byte fixed width encoding.

12.3.7.1.2 Text Length in Characters when Specified in Bytes

If a simple element has dfdl:representation='text' but dfdl:lengthUnits='bytes' then the following rules apply:

The length of the simple content region (i.e., SimpleContent defined in Section **Error! Reference source not found.**)

- is the length in bytes, the length being calculated using the normal rules.
- When parsing, as many characters as possible are extracted from the bytes of the simple content region. Any left over bytes are skipped.
- When unparsing, if the simple content region is larger than the encoded length of the element then the remaining bytes are filled with dfdl:fillByte (This is the grammar ***RightPadOrFill*** region.).

Comment [m260]: 2.100 (DONE)

Added this section to clarify that 'explicit' and 'prefixed' are essentially variable length when unparsing.

Please review.

SMH: Corrected. But not happy that this is so far away from the initial section on parsing specified lengths, which also talks about unparsing. I think this should all be in the initial section.

Fixed. Now in 12.3.7 which is the first discussion of specified length.

Comment [m261]: DONE – align colums of header row with data row in this table.

Comment [Mike Beck262]: Errata 2.6 DONE

12.3.7.1.3 Byte Order Mark

DFDL only interprets Unicode byte order marks at the document level, that is, at the start of the document information item, and the start of the data stream. See Section 11.1 Unicode Byte Order Marks (BOM) for details.

If a byte-order mark codepoint appears within the data, either within or at the start of a UTF-8, UTF-16 or UTF-32 encoded string then the byte-order mark will be included as part of the string payload[22] as a character codepoint. That is, for the UTF-8, UTF-16 and UTF-32 character encodings, a byte-order-mark codepoint is treated as a character of the string in DFDL and contributes 1 character length unit to the length.

A way of eliminating the byte-order mark at the beginning of a string so that it does not end up in the infoset is that the byte-order mark can be modeled as a separate element before the string. This BOM element can be either required or optional depending on whether a byte order mark is expected or optional at the beginning of the string.

> **Comment [SMH263]:** Errata 3.7: Needs revising, could just be a reference to new words in section 11?
>
> DONE

> **Comment [m264]:** Errata 2.90 DONE

12.3.7.2 Length of Bit Fields

A bit field is an element of signed or unsigned integer or boolean type where the lengthUnits='bits'.

For signed integer types, the bits are interpreted as a twos-complement integer with the most-significant bit representing the sign (0 for positive, 1 for negative).

It is a schema definition error if the length of a bit field is too large for the corresponding integer type, and the length can be statically determined (from the schema only).

It is a schema definition error if the type is a signed integer type, and the length is 1 bit.

It is a runtime schema definition error if the length of a bit field can only be determined dynamically (for example because it is stored in the data), and the resulting length is too large for the integer type, or is 1 bit for a signed integer type.

Definition: Bit position

The data stream is assumed to be a collection of consecutively numbered unsigned bytes. Each byte is a numeric value, and bit position within an individual byte is given by numeric behavior. The bits within each byte are numbered, with the most significant bit having position 1, and the least significant bit having position 8.

This gives every bit in the data stream a specific bit position. Furthermore, the bit position of the least significant bit of byte N is numerically adjacent to the bit position of the most significant bit of byte N+1.

Definition: Bit string

For types xs:boolean, xs:byte, xs:short, xs:int, xs:long, xs:unsignedByte, xs:unsignedShort, xs:unsignedInt, and xs:unsignedLong, it is possible to specify dfdl:lengthUnits='bits', dfdl:lengthKind='explicit', and then provide a dfdl:length expression. This expression must be a literal integer, the value of which is between (inclusively) 1 and 32, 8, 16, 32, and 64 respectively for unsigned types, and between 2 and those same upper limits for the corresponding signed types. Such an element is called a bit string for brevity.

> **Comment [m265]:** Errata 2.90 Done

If the dfdl:length property contains a value out of range then it is a schema definition error. Note that for signed number types, the minimum value for the dfdl:length is 2.

> **Comment [m266]:** Errata 2.90 as revised. DONE

---

[22] The codepoint of the byte-order-mark is the same as the ZWNBS (Zero-Width Non-Breaking Space) character.

When parsing, if the data stream ends without enough bits to parse a bit string, that is, N bits are needed based on the dfdl:length, but only M < N bits are available, then it is a processing error.

(Note: This is not specific to bit strings. Any type whose length cannot be satisfied from the data will cause a processing error.

Bit strings, Alignment, and dfdl:fillByte

The dfdl:alignmentUnits='bits', and dfdl:alignment='1' can be used to position a bit string anywhere in the data stream without regard for any other grouping of bits into bytes.

The numeric value of the unsigned integer represented by a bit string is unaffected by alignment.

When unparsing a bit string, alignment may cause the bits of the bit string to occupy only some of the bits within a byte of the data stream. The bits of data in the alignment fill region are unspecified by the elements of the DFDL schema, and are not found in the DFDL infoset. Such unspecified bits are filled in using the value of the dfdl:fillByte property. Corresponding bits from the dfdl:fillByte value are used to fill in unspecified bits of the data stream. That is, if bit K (K will be 1 or greater, but less than or equal to 8) of a data stream byte is unspecified, its value will be taken from bit K of the dfdl:fillByte property value.

Since the value of any bit string element is unaffected by alignment, the logical integer value for a bit-string is always computed as if the first bit were at position 1 of the bit stream. If the dfdl:length for the bit-string evaluates to M, then the bit-string conceptually occupies bits 1 to M of a data stream for purposes of computing its value.

Bits within Bit Strings of Length <= 8

Any time the length in bits is < 8, then when set, the bit at position Z supplies value $2^{(M-Z)}$, and the value of the bit string as an integer is the sum of these values for each of its bits.

Bits within Bit Strings of Length > 8

Call M the length of the bit string element in bits. In general, when M > 8 the contribution of a bit in position i to the numeric value of a bit string is given by a formula specific to the dfdl:byteOrder.

For dfdl:byteOrder='bigEndian' the value of bit i is given by $2^{(M - i)}$.

For dfdl:byteOrder='littleEndian' the value of bit i is given by a more complex formula. The following pseudo code computes the value of a bit in a littleEndian bit string. It is just a very big expression, but is spread out over many local variables to illustrate the various sub-calculations clearly. DFDL implementations may use any way of converting bit strings to the corresponding integer values that is consistent with this:

In the pseudo code below:

- '%' is modular division (division where remainder is returned)

- '/' is regular division (quotient is returned)

- the expression 'a ? b : c' means 'if a is true, then the value is b, otherwise the value is c'

```
    littleEndianBitValue(bitPosition, bitStringLength)
        assert bitPosition >= 1;
        assert bitStringLength >= 1;
        assert bitStringLength >= bitPosition;
        numBitsInFinalPartialByte = bitStringLength % 8;
        numBitsInWholeBytes = bitStringLength –
                            numBitsInFinalPartialByte;
        bitPosInByte = ((bitPosition – 1) % 8) + 1;
        widthOfActiveBitsInByte = (bitPosition <= numBitsInWholeBytes)
            ? 8 : numBitsInFinalPartialByte;
        placeValueExponentOfBitInByte = widthOfActiveBitsInByte –
                                    bitPosInByte;
```

```
        bitValueInByte = 2^placeValueExponentOfBitInByte;
        byteNumZeroBased = (bitPosition – 1)/8;
        scaleFactorForBytePosition = 2^(8 * byteNumZeroBased);
        bitValue = bitValueInByte * scaleFactorForBytePosition;
        return bitValue;
```

Figure 4 Little Endian bit position and value

Examples

Consider the first three bytes of the data stream. Imagine their numeric values as 0x5A 0x92 0x00.

```
Positions:
00000000 01111111 11122222
12345678 90123456 78901234
Bits:
01011010 10010010 00000000

Hex values
   5   A   9   2   0   0
```

Beginning at bit position 1, (the very first bit) if we consider the first two bytes as a bigEndian short, the value will be 0x5A92.

```
  <xs:element name="num" type="unsignedShort"
        dfdl:alignment="1"
        dfdl:alignmentUnits="bytes"
        dfdl:byteOrder="bigEndian"
        dfdl:representation="binary"/>
```

As a littleEndian short, the value will be 0x925A.

```
  <xs:element name="num" type="unsignedShort"
        dfdl:alignment="1"
        dfdl:alignmentUnits="bytes"
        dfdl:byteOrder="littleEndian"
        dfdl:representation="binary"/>
```

Now let us examine a bit string of length 13, beginning at position 2.

```
<xs:sequence>
  <xs:element name="ignored" type="unsignedByte"
        dfdl:alignment="1"
        dfdl:alignmentUnits="bits"
        dfdl:lengthUnits="bits"
        dfdl:length="1"
        dfdl:representation="binary"/>
  <xs:element name="x" type="unsignedShort"
        dfdl:alignment="1"
        dfdl:alignmentUnits="bits"
        dfdl:byteOrder="bigEndian"
        dfdl:lengthUnits="bits"
        dfdl:length="13"
        dfdl:representation="binary"/>
   ...
</xs:sequence>
```

Let's examine the same data stream and consider the bit positions that make up element 'x', which are the bits at positions 2 through 14 inclusive.

```
Positions:
00000000 01111111 11122222
```

```
12345678 90123456 78901234
Bits:
 1011010 100100
```

Since alignment does not affect logical value, we will obtain the same logical value as if we realigned the bits. That is, the value is the same as if we began the bits of the element's representation with bit position 1.

```
Realigned Positions:
00000000 01111111 11122222
12345678 90123456 78901234
Bits:
10110101 00100
```

The DFDL schema fragment above gives element 'x' the dfdl:byteOrder='bigEndian' property. In this case the place value of each position is given by $2^{(M – i)}$

```
PlaceValue positions 2^(M - i)
```

```
...11110 00000000
...21098 76543210
```

```
Bit values
```

```
...10110 10100100
Hex values
  1   6    A   4
```

The value of element 'x' is 0x16A4. Notice how it is the most-significant byte -- which is the first byte when big endian -- that becomes the partial byte (having fewer than 8 bits) in the case where the length of the bit string is not a multiple of 8 bits.

For dfdl:byteOrder='littleEndian'. The place values of the individual bits are not as easily visualized. However there is still a basic formula (given in the pseudo code in Figure 4 Little Endian bit position and value.

Looking again at our realigned positions:

```
Realigned Positions:
00000000 01111111 11122222
12345678 90123456 78901234
Bits:
10110101 00100
```

The place values of each of these bits, for little endian byte order can be seen to be:

```
PlaceValue positions
```

```
00000000 ...11100
76543210 ...21098
```

```
Bit values
```

```
10110101 ...00100
Hex values
  B   5    0   4
```

We must reorder the bytes for little endian byte order. The value of element 'x' is 0x04B5. In little endian form, the first 8 bits make up the first byte, and that contains the least-significant byte of the logical numeric unsignedShort value. The additional bits of the partial byte are once again the most significant byte; however, for little endian form, this is the second byte. The second byte contains only 5 bits, those make up the least significant 5 bits of that byte, but that logical 5-bit value makes up the most-significant byte of the unsignedShort integer.

Booleans

The properties dfdl:binaryBooleanTrueRep and dfdl:binaryBooleanFalseRep are unsigned integers. Specifically, their numeric ranges are restricted as if of type xs:unsignedInt, with additional restriction in range when the dfdl:lengthUnits='bits' and dfdl:length are used to specify fewer than the maximum of 32 bits.

12.3.7.3 Length of Complex Element of Specified Length

A complex element of known length is defining a 'box' in which its child element exist. For example a fixed length record with a variable number of children that may not fill the full length of the record.

For example, an element of complex type may have explicit length of 100 bytes, but contain a sequence of child elements that use up less than 100 bytes of data. In this case the remaining unused data is called the *ElementUnused*. It is skipped when parsing, and is filled with the dfdl:fillByte on unparsing.

> **Comment [SMH267]:** Errata 2.55. **DONE**

When the dfdl:lengthUnits is 'characters' on a complex element of specified length then the dfdl:lengthUnits of all its children must be characters also with the same encoding, otherwise it is a schema definition error.

12.3.8 Summary: Length of Simple Types with Binary Representations

Elements of type xs:hexBinary or with dfdl:representation 'binary' and a packed decimal representation may be delimited, endOfParent or specified length. All other elements with dfdl:representation 'binary' must be implicit or specified length and it is a schema definition error if dfdl:lengthKind is 'delimited', or 'endOfParent' where the parent dfdl:lengthKind is 'delimited'.

> **Comment [SMH268]:** Isn't this just restating earlier rules?
>
> Yes, but it's worth resummarizing, and including pattern as well.
>
> (DONE)
>
> Added pattern, and endOfData, which had been skipped. Added hexbinary.
>
> SMH: Pattern removed – not supported. HexBinary can't be pattern either.

It is a schema definition error if an element with binary representation has dfdl:lengthKind='pattern'.

13. Simple Types

The 'representation' property identifies the physical representation of the element. The DFDL logical types are grouped to illustrate which physical representations apply to each logical type.

These properties provide the correct interpretation of the data found in the SimpleContent grammar region.

The allowable physical representations for each logical type grouping are also shown, where the logical type groupings are defined as:

Logical Type Group	Types
Number	xs:double, xs:float, xs:decimal, xs:integer and its restrictions (xs:int, xs:unsignedLong, etc.)
String	xs:string
Calendar	xs:dateTime, xs:date, xs:time
Opaque	xs:hexBinary
Boolean	xs:boolean

Table 17 Logical type groups

13.1   Properties Common to All Simple Types

Property Name	Description
representation	Enum Valid values are dependent on logical type. **Number:** 'text, 'binary' **String:** representation is assumed to be 'text' and the representation property is not examined **Calendar:** 'text, 'binary' **Boolean:** 'text, 'binary' **Opaque:** representation is assumed to be 'binary' and the representation property is not examined. Annotation: dfdl:element, dfdl:simpleType

The permitted representation properties for each logical type are shown in Table 18: Logical Type to Representation properties

Logical type	dfdl:representation	Additional representation property
String	Assumed to be text	
Float, Double	text	**textNumberRep:** standard
	binary	**binaryFloatRep**: ieee, ibm390Hex
Decimal, Integer, nonNegativeInteger	text	**textNumberRep:** standard, zoned
	binary	**binaryNumberRep:** packed, bcd, ibm4690Packed, binary

> **Comment [SMH269]:** Not a terminal region, so normal font.
>
> DONE

Long, Int, Short, Byte, UnsignedLong, Unsignedint, Unsignedshort, UnsignedByte	text	**textNumberRep:** standard, zoned
	binary	**binaryNumberRep:** packed, bcd, ibm4690Packed, binary
DateTime, Date, Time	text	
	binary	**binaryCalendarRep:** packed, bcd, ibm4690Packed, binarySeconds, binaryMilliseconds
Boolean	text	
	binary	
HexBinary	Assumed to be binary	

> **Comment [SMH270]:** Strictly, bcd is not allowed for Long, Int, Short, Byte as it can't rep a negative number. So should really split the number row in two.
>
> DONE (Decided not to split.)

Table 18: Logical Type to Representation properties

13.2   Properties Common to All Simple Types with Text representation

Property Name	Description
textPadKind	Enum
	Valid values 'none', 'padChar'.
	Indicates whether to pad the representation text on unparsing.
	'none': No padding occurs. When lengthKind is 'implicit' or 'explicit' the representation text must match the expected length otherwise it is a processing error.
	'padChar': The element is padded using the dfdl:textStringPadCharacter, dfdl:textNumberPadCharacter, dfdl:textBooleanPadCharacter or dfdl:textCalendarPadCharacter depending on the type of the element
	When lengthKind is 'implicit' the element is padded to the implicit length for the type.
	When lengthKind is 'explicit' the element is padded to the length given by the dfdl:length property.
	When lengthKind is 'delimited', 'prefixed', 'pattern' or 'endOfParent' the element is padded to the length given by the xs:minLength facet for type 'xs:string' or dfdl:textOutputMinLength property for other types.
	Annotation: dfdl:element, dfdl:simpleType
textTrimKind	Enum
	Valid values 'none', 'padChar'
	Indicates whether to trim data on parsing.
	When 'none' no trimming takes place
	When 'padChar' the element is trimmed of the dfdl:textStringPadCharacter, dfdl:textNumberPadCharacter, dfdl:textBooleanPadCharacter or dfdl:textCalendarPadCharacter depending on the type of the element..
	Annotation: dfdl:element , dfdl:simpleType

textOutputMinLength	Non-negative Integer.
	Only used when dfdl:textPadKind is 'padChar' and dfdl:lengthKind is 'delimited', 'prefixed', 'pattern', 'explicit' (when dfdl:length is an expression) or 'endOfParent', and type is not xs:string
	Specifies the minimum representation length during unparsing for simple types that do not allow the xs:minLength facet to be specified. The units are specified by the dfdl:lengthUnits property.
	If dfdl:textOutputMinLength is zero or less than the length of the representation text then no padding occurs.
	Annotation: dfdl:element, dfdl:simpleType
escapeSchemeRef	QName or empty String
	The name of the dfdl:defineEscapeScheme annotation that provides the additional properties used to describe the escape scheme. If the value is the empty string then escaping is explicitly turned off.
	See: The dfdl:defineEscapeScheme Defining Annotation Element and The dfdl:escapeScheme Annotation Element
	Annotation: dfdl:element, dfdl:simpleType

Comment [m271]: 2.100 (DONE)

Added this. Please review.

SMH: Looks good.

13.2.1 The dfdl:escapeScheme Properties

The dfdl:escapeScheme annotation is used within a dfdl:defineEscapeScheme annotation to group the properties of an escape scheme and allows a common set of attributes to be defined that can be reused.

An escape scheme is needed when the content of a text element contains sequences of characters that are the same as an in-scope separator or terminator. If the characters are not escaped, a parser scanning for a separator or terminator would erroneously find the character sequence in the content.

An escape scheme defines the properties that describe the text escaping rules. There are two variants on such schemes:

- The use of a single escape character to cause the next character to be interpreted literally. The escape character itself is escaped by the escape escape character.

- The use of a pair of escape strings to cause the enclosed group of characters to be interpreted literally. The ending escape string is escaped by the escape escape character.

On parsing, the escape scheme is applied after padding characters are trimmed and on unparsing before padding characters are added.

On unparsing, the application of escape scheme processing takes place before the application of the emptyValueDelimiterPolicy property.

Comment [m272]: Errata 2.74 DONE

DFDL does not provide a substitution mechanism similar to XML that would replace a character entity such as < with its literal value <.

Property Name	Description
escapeKind	Enum
	Valid values 'escapeCharacter', 'escapeBlock'

	The type of escape mechanism defined in the escape scheme
	When 'escapeCharacter': On unparsing a single character of the data is escaped by adding an dfdl:escapeCharacter before it. The following are escaped if they are in the data
	• Any in-scope terminating delimiter by escaping its first character.
	• dfdl:escapeCharacter (escaped by dfdl:escapeEscapeCharacter)
	• any dfdl:extraEscapedCharacters
	On parsing the dfdl:escapeCharacter and dfdl:escapeEscapeCharacter are removed from the data, unless the dfdl:escapeCharacter is preceded by the dfdl:escapeEscapeCharacter, or the dfdl:escapeEscapeCharacter does not proceed the dfdl:escapeCharacter.
	When 'escapeBlock': On unparsing the entire data are escaped by adding dfdl:escapeBlockStart to the beginning and dfdl:escapeBlockEnd to the end of the data. The data is either always escaped or escaped when needed as specified by dfdl:generateEscapeBlock. If the data is escaped and contains the dfdl:escapeBlockEnd then first character of each appearance of the dfdl:escapeBlockEnd is escaped by the dfdl:escapeEscapeCharacter.
	On parsing the dfdl:escapeBlockStart is removed from the beginning of the data and dfdl:escapeBlockEnd is removed from end of the data and any dfdl:escapeEscapeCharacters are removed when they precede a dfdl:escapeBlockEnd.
	Annotation: dfdl:escapeScheme
escapeCharacter	DFDL String Literal or DFDL Expression
	Specifies one character that escapes the subsequent character.
	Used when dfdl:escapeKind = 'escapeCharacter'
	It is a schema definition error if dfdl:escapeCharacter is empty when dfdl:escapeKind is 'escapeCharacter'
	This property can be computed by way of an expression which returns a character. The expression must not contain forward references to elements which have not yet been processed.
	Escape and Quoting Character Restrictions: The string literal is restricted to allow only certain kinds of syntax:
	• DFDL character entities are allowed
	• The raw byte entity (%#r) is not allowed
	• DFDL Character classes (NL, WSP, WSP+, WSP*, ES) are not allowed
	It is a schema definition error if the string literal contains any of the disallowed constructs.
	Escape characters contribute to the representation length of the field
	Annotation: dfdl:escapeScheme
escapeBlockStart	DFDL String Literal
	The string of characters that denotes the beginning of a sequence of

Comment [SMH273]: Errata 2.16 clarifies allowable string literal content

Comment [m274]: Done

Comment [SMH275]: Errata 2.16 clarifies allowable string literal content
DONE

Comment [m276]: Done

	characters escaped by a pair of escape strings.
	Used when dfdl:escapeKind = 'escapeBlock'
	It is a schema definition error if escapeBlockStart is empty when dfdl:escapeKind is 'escapeBlock'
	The string literal value is restricted in the same way as described in "*Escape and Quoting Character Restrictions*" in the description of the escapeCharacter property.
	An dfdl:escapeBlockStart string contributes to the representation length of the field
	Annotation: dfdl:escapeScheme
escapeBlockEnd	DFDL String Literal
	The string of characters that denotes the end of a sequence of characters escaped by a pair of escape strings.
	Used when dfdl:escapeKind = 'escapeBlock' .
	It is a schema definition error if dfdl:escapeBlockEnd is empty when dfdl:escapeKind is 'escapeBlock'
	The string literal value is restricted in the same way as described in "*Escape and Quoting Character Restrictions*" in the description of the escapeCharacter property.
	A dfdl:escapeBlockEnd string contributes to the representation length of the field
	Annotation: dfdl:escapeScheme
escapeEscapeCharacter	DFDL String Literal or DFDL Expression
	Specifies one character that escapes the subsequent escape character or first character of dfdl:escapeBlockEnd.
	Used when dfdl:escapeKind = 'escapeCharacter' or 'escapeBlock'.
	This property can be computed by way of an expression which returns a character. The expression must not contain forward references to elements which have not yet been processed.
	The string literal value is restricted in the same way as described in "*Escape and Quoting Character Restrictions*" in the description of the escapeCharacter property.
	If the empty string is specified then no escaping of escape characters occurs.
	It is explicitly allowed for both the dfdl:escapeCharacter and the dfdl:escapeEscapeCharacter to be the same character. In that case processing functions as if the dfdl:escapeCharacter escapes itself.
	Annotation: dfdl:escapeScheme
extraEscapedCharacters	List of DFDL String Literals
	A space separated list of single characters that must be escaped in addition to the in-scope delimiters. If there are no extra characters to escape the property should be set to "".
	The string literal values are restricted in the same way as described in "*Escape and Quoting Character Restrictions*" in the description of the

Comment [SMH277]: Insert paragraph break
DONE

Comment [SMH278]: Errata 2.16 clarifies allowable string literal content
DONE
Comment [m279]: Done

Comment [SMH280]: Errata 2.16 clarifies allowable string literal content
Comment [m281]: Done

	escapeCharacter property.
	This property only applies on unparsing.
	Annotation: dfdl:escapeScheme
generateEscapeBlock	Enum
	Valid values 'always', 'whenNeeded'
	Controls when escaping is used on unparsing when dfdl:escapeKind is 'escapeBlock'.
	If 'always' then escaping is always occurs as described in dfdl:escapeKind.
	If 'whenNeeded' then escaping occurs as described in dfdl:escapeKind when the data contains any of the following:
	• any in-scope terminating delimiter
	• dfdl:escapeBlockStart at the start of the data
	• any dfdl:extraEscapedCharacters
	Annotation: dfdl:escapeScheme

13.3   Properties for Bidirectional support for All Simple Types with Text representation

Bidirectional text consists of mainly right-to-left text with some left-to-right nested segments (such as an Arabic text with some information in English), or vice versa (such as an English letter with a Hebrew address nested within it.)

Note: the bidirectional properties apply to the content of the element and not to the initiator, terminator or separator if defined.

Property name	Description
textBidi	Enum Valid values are 'yes', 'no' Indicates the text content of the element is bidirectional. Annotation: dfdl:element, dfdl:simpleType (representation text)
textBidiOrdering	Enum Valid values 'implicit', 'visual'. Defines how bidirectional text is stored in memory. 'Implicit' means that the characters are stored in the order they are read or typed. That is with the first character in the first position in the data. (This is also called logical). 'Visual means that the characters are stored in the order they would be printed or displayed. That is, the last character of a right to left sequence is in the first position in the data and the first character of a left to right sequence is in the first position in the data. Annotation: dfdl:element , dfdl:simpleType (representation text) ,
textBidiOrientation	Enum Valid values 'LTR', 'RTL', 'contextual_LTR', 'contextual_RTL'. Indicates how the text should be displayed. 'LTR' means left-to-right 'RTL' mean right to left. 'contextual_LTR' and 'contextual_RTL' means that the orientation should be taken from the context of the data. The data may contain 'strong' characters that are either orientation left or orientation right. The term following contextual (LTR or RTL) specifies what should be the default orientation when the data are orientation-neutral (i.e. there are no strong characters). Annotation: dfdl:element, dfdl:simpleType (representation text)
textBidiSymmetric	Enum Valid values are 'yes', 'no' Defines whether characters such as < ([{ that have a symmetric character with an opposite directional meaning: >)] } should be swapped Annotation: dfdl:element, dfdl:simpleType (representation text)
textBidiShaped	Enum Valid values are 'yes', 'no' Defines whether characters should be shaped on unparsing. Character

Comment [SMH282]: Errata 2.60 changes name. **DONE**

Comment [SMH283]: Errata 2.60 changes name **DONE**

	shaping occurs when the shape of a character is dependent on its position in a word.
	Annotation: dfdl:element, dfdl:simpleType (representation text)
textBidiNumeralShapes	Enum
	Valid values 'nominal', 'national'.
	Defines on unparsing whether logical numbers with text representation should have Arabic shapes (0123456789) or Arabic-Indic (٠١٢٣٤٥٦٧٨٩)
	When 'nominal': All numbers are presented using Arabic shapes
	When 'national': All numbers are presented using Arabic-Indic shapes.
	Annotation: dfdl:element, dfdl:simpleType (number with representation text)

13.4   Properties Specific to Strings with Text representation

Property Name	Description
textStringJustification	Enum
	Valid values 'left', 'right', 'center'
	Unparsing:
	'left': Justifies to the left and adds padding chars to the string contents if the string is too short, to the length determined by the dfdl:textPadKind property.
	'right': Justifies to the right and adds padding chars to the string contents if the string is too short, to the length determined by the dfdl:textPadKind property.
	'center': Adds equal padding chars left and right of the string contents if the string is too short, to the length determined by the dfdl:textPadKind property. It adds one extra padding char on the left if needed.
	Parsing:
	'left': Trims any padding characters from the right of the string, according to dfdl:textTrimKind property.
	'right': Trims any padding characters from the left of the string, according to dfdl:textTrimKind property.
	'center' Trims any padding characters from the left and right of the string, according to dfdl:textTrimKind property.
	Annotation: dfdl:element, dfdl:simpleType
textStringPadCharacter	DFDL String Literal
	The value that is used when padding or trimming string elements. The value can be a single character or a single byte.
	If a character, then it can be specified using a literal character or using DFDL entities.

Comment [SMH284]: Errata 2.17 clarifies allowable string literal content
DONE

	If a byte, then it must be specified using a single byte value entity otherwise it is a schema definition error
	If a pad character is specified when dfdl:lengthUnits='bytes' then the pad character must be a single-byte character.
	If a pad byte is specified when dfdl:lengthUnits='characters' then - the encoding must be a fixed-width encoding - padding and trimming must be applied using a sequence of N pad bytes, where N is the width of a character in the fixed-width encoding.

	Padding *Character Restrictions:* The string literal is restricted to allow only certain kinds of syntax:
	
	   • DFDL character entities are allowed
	   • The raw byte entity (%#r) is allowed.
	   • DFDL Character classes (NL, WSP, WSP+, WSP*, ES) are not allowed
	
	It is a schema definition error if the string literal contains any of the disallowed syntax.

	Annotation: dfdl:element, dfdl:simpleType
truncateSpecifiedLengthString	Enum
	Valid values are 'yes', 'no'
	Used on unparsing only
	'yes' means if the logical type is xs:string and the value is longer than the specified length, the string is truncated to this length. (See section 12.3.7 Elements of Specified Length.) No processing error is raised.
	The position from which data is truncated is determined by the value of the dfdl:textStringJustification property. If the value of the dfdl:textStringJustification property is 'left', data is truncated from the right; if the value of the dfdl:textStringJustification property is 'right', data is truncated from the left. However if the value of the dfdl:textStringJustification property is 'center', truncation does not occur and a processing error occurs if the value is too long.
	When unparsing, validation errors cannot be prevented by truncation as validation takes place on the augmented infoset, before any truncation has occurred.
	Annotation: dfdl:element, dfdl:simpleType

Comment [SMH285]: 'PaddingCharacter Restrictons'
DONE

Comment [m286]: Errata 2.17 Done

Comment [SMH287]: Add full stop
DONE

Comment [SMH288]: Errata 3.12 changes unparsing validation so that it takes place on the augmented Infoset. The setting of this property therefore can not prevent validation errors. I think this is worth stating.

DONE

Comment [SMH289]: Errata 3.12 changes how validation is defined, we should state here the interaction with this property.

DONE

13.5   Properties Specific to Number with Text or Binary representation

Property Name	Description
decimalSigned	Enum
	Valid values are 'yes', 'no'
	Indicates whether an xs:decimal element is signed. See 13.6.2 Converting logical numbers to/from text representation and 13.7.1 Converting Logical Numbers to/from Binary to see how this affects the presence of the sign in the data stream.
	'yes' means that the xs:decimal element is signed
	'no' means that the xs:decimal element is not signed
	Annotation: dfdl:element, dfdl:simpleType

13.6   Properties Specific to Number with Text representation

Property Name	Description
textNumberRep	Enum
	Valid values are 'standard', 'zoned'
	'standard' means represented as characters in the character set encoding specified by the dfdl:encoding property.
	'zoned' means represented as a zoned decimal in the character set encoding specified by the dfdl:encoding property. Zoned is not supported for float and double numbers. Base 10 is assumed, and the encoding must be for a single-byte character set. It is a schema definition error if any of these requirements are not met.
	Annotation: dfdl:element, dfdl:simpleType
textNumberJustification	Enum
	Valid values 'left', 'right', 'center'
	Controls how the data is padded or trimmed on parsing and unparsing.
	Behavior as for dfdl:textStringJustification.
	Annotation: dfdl:element, dfdl:simpleType
textNumberPadCharacter	DFDL String Literal
	The value that is used when padding or trimming number elements.
	The value can be a single character or a single byte.
	If a character, then it can be specified using a literal character or using DFDL entities.
If a byte, then it must be specified using a single byte value entity
	If a pad character is specified when dfdl:lengthUnits='bytes'

Comment [m290]: **Errata 2.93**. *Sections 13.6, 13.7.* DONE.

Comment [m291]: **Errata 2.106.** DONE

Comment [SMH292]: Errata 2.15. **DONE**

Comment [m293]: Errata 2.92 DONE

Comment [SMH294]: Errata 2.17 clarifies allowable string literal content

Comment [m295]: DONE

	then the pad character must be a single-byte character.
	If a pad byte is specified when dfdl:lengthUnits='characters' then
	• the encoding must be a fixed-width encoding
	• padding and trimming must be applied using a sequence of N pad bytes, where N is the width of a character in the fixed-width encoding.
	When trimming, the last remaining digit is never trimmed for text numbers regardless of its value (so even if the textNumberPadCharacter is 0, data containing 0000 will be trimmed to 0, not empty string.
	The string literal value is restricted in the same way as described in "Padding Character Restrictions" in the description of the textStringPadCharacter property.
	Annotation: dfdl:element, dfdl:simpleType
textNumberPattern	String
	Defines the ICU-like pattern that describes the format of the text number. The pattern defines where grouping separators, decimal separators, implied decimal points, exponents, positive signs and negative signs appear. It permits definition by either digits/fractions or significant digits. Allows rounding.
	When dfdl:textNumberRep is 'standard' this property only applies when dfdl:textStandardBase is 10. When dfdl:textNumberRep is 'standard' and dfdl:textStandardBase is not 10 the number is represented as the minimum number of characters to represent the digits. There is no sign or virtual decimal point.
	The syntax of dfdl:textNumberPattern is described in section 13.6.1 The textNumberPattern Property
	Annotation: dfdl:element, dfdl:simpleType
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 the rounding increment is specified in the dfdl:textNumberPattern using digits '1' though '9'. The rounding mode is always 'roundHalfEven'. To switch off rounding, do not use digits '1' through '9'.
	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 switch off rounding, use 0.0 for the dfdl:textNumberRoundingIncrement.

Comment [m296]: Errata 2.25 Done

Comment [SMH297]: Padding Character Restrictions DONE

Comment [m298]: Errata 2.17 Done

Comment [SMH299]: I think this is meant to be errata 2.17 but I can only see boxes. It should be referring to Padding Character Restrictions in property textStringPadCharacter.

FIXED DONE.

	When unparsing a number and excess precision is supplied in the Infoset and rounding is not in effect, it is 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.
	Valid values 'roundCeiling', 'roundFloor', 'roundDown', 'roundUp', 'roundHalfEven', 'roundHalfDown', 'roundHalfUp'
	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.
	To switch off rounding, use 0.0.
	When unparsing a number and excess precision is supplied in the Infoset and rounding is not in effect, it is a processing error.
	A negative value is a schema definition error.
	Annotation: dfdl:element, dfdl:simpleType
textNumberCheckPolicy	Enum
	Values are 'strict' and 'lax'.
	Indicates how lenient to be when parsing against the pattern.
	When dfdl:textNumberRep is 'standard' this property only applies when dfdl:textStandardBase is 10.
	If 'lax' and dfdl:textNumberRep is 'standard' then grouping separators are ignored, leading and trailing whitespace is ignored, leading zeros are ignored and quoted characters may be omitted.
	If 'lax' and dfdl:textNumberRep is 'zoned' then positive punched data is accepted when parsing an unsigned type, and unpunched data is accepted when parsing a signed type
	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.
	If 'strict' and dfdl:textNumberRep is 'zoned' then the data must follow the pattern.
	On unparsing the pattern is always followed and follow the rules in 13.6.2 Converting logical numbers to/from text representation.

Comment [m300]: Errata 2.93 DONE

Comment [SMH301]: Needs some more thought, so we define lax behaviour followed by what strict buys on top.

Comment [m302]: (DONE)

SMH: Action 170 means we can close this for now. OK.

Comment [SMH303]: Errata 2.21 changes this to be consistent with ICU

DONE

Comment [m305]: Errata 2.21 and 2.54 DONE

Comment [SMH304]: Updated 2.54 DONE

	Annotation: dfdl:element, dfdl:simpleType
textStandardDecimalSeparator	List of DFDL String Literals or DFDL Expression
	Defines a whitespace separated list of single characters that appear (individually) in the data as the decimal separator.
	This property is applicable when type is xs:decimal, xs:float or xs:double and dfdl:textNumberRep is 'standard' and dfdl:textStandardBase is 10. It must be set if dfdl:textNumberPattern contains a decimal separator symbol ("."), or the E or @ symbols (it is a schema definition error otherwise.) Empty string is not an allowable value.
	This property can be computed by way of an expression which returns a character. The expression must not contain forward references to elements which have not yet been processed.
	Text Number Character Restrictions: The the string literal is restricted to allow only certain kinds of syntax:
	• DFDL character entities are allowed
	• The raw byte entity (%#r) is not allowed.
	• DFDL Character classes (NL, WSP, WSP+, WSP*, ES) are not allowed
	It is a schema definition error if the string literal contains any of the disallowed syntax.
	Annotation: dfdl:element, dfdl:simpleType
textStandardGroupingSeparator	DFDL String Literal or DFDL Expression
	Defines the single character that will appear in the data as the grouping separator.
	This property is applicable when dfdl:textNumberRep is 'standard' and dfdl:textStandardBase is 10. It must be set if dfdl:textNumberPattern contains a grouping separator symbol (it is a schema definition error otherwise.) Empty string is not an allowable value.
	This property can be computed by way of an expression which returns a character. The expression must not contain forward references to elements which have not yet been processed.
	The string literal value is restricted in the same way as described in "*Text Number Character Restrictions*" in the description of the dfdl:textStandardDecimalSeparator property.
	Annotation: dfdl:element, dfdl:simpleType
textStandardExponentRep	DFDL String Literal or DFDL Expression
	Defines the actual character(s) that will appear in the data as the exponent indicator. If the empty string is specified then no exponent character will be used.
	This property is applicable when dfdl:textNumberRep is

Comment [SMH306]: Errata 2.18 clarifies allowable string literal content DONE

Comment [SMH307]: Errata 2.23 allows a list of sring literals DONE

Comment [SMH308]: Errata 2.27: **DONE**

Comment [SMH309]: Add full stop DONE

Comment [SMH310]: Errata 2.22: **DONE**

Comment [SMH311]: Text Number Character Restrictions.

DONE

Comment [m312]: Errata 2.18 Done

Comment [SMH313]: Delete DONE

Comment [SMH314]: Errata 2.18 clarifies allowable string literal content DONE

Comment [SMH315]: Add full stop DONE

Comment [SMH316]: Errata 2.24: **DONE**

Comment [SMH317]: Text Number Character Restrictions DONE

Comment [m318]: Errata 2.18 Done

Comment [SMH319]: Errata 2.46 changes name. **DONE**

Comment [SMH320]: Errata 2.18 clarifies allowable string literal content DONE

Comment [SMH321]: Errata 2.46 allows multiple characters **DONE**

	'standard' and dfdl:textStandardBase is 10. Empty string is an allowable value, so that formats like NNN+M (meaning NNN x 10 with MM exponent) can be expressed.
	This property must be set even if the dfdl:textNumberPattern does not contain an 'E' (exponent) character. It is a schema definition error if this property is not set or in scope for any number with dfdl:representation='text'.
	This property can be computed by way of an expression which returns a DFDL String Literal. The expression must not contain forward references to elements which have not yet been processed.
	The string literal value is restricted in the same way as described in "*Text Number Character Restrictions*" in the description of the dfdl:textStandardDecimalSeparator property.
	If dfdl:ignoreCase is 'yes' then the case of the string is ignored by the parser.
	Annotation: dfdl:element, dfdl:simpleType
textStandardInfinityRep	DFDL String Literal
	The value used to represent infinity.
	Infinity is represented as a string with the positive or negative prefixes and suffixes from the dfdl:textNumberPattern applied.
	This property is applicable when dfdl:textNumberRep is 'standard', dfdl:textStandardBase is 10 and the simple type is float or double.
	If dfdl:ignoreCase is 'yes' then the case of the string is ignored by the parser.
	The string literal value is restricted in the same way as described in "*Text Number Character Restrictions*" in the description of the dfdl:textStandardDecimalSeparator property.
	Annotation: dfdl:element, dfdl:simpleType
textStandardNaNRep	DFDL String Literal
	The value used to represent NaN.
	NaN is represented as string and the positive or negative prefixes and suffixes from the dfdl:textNumberPattern are not used.
	This property is applicable when dfdl:textNumberRep is 'standard', dfdl:textStandardBase is 10 and the simple type is float or double.
	If dfdl:ignoreCase is 'yes' then the case of the string is ignored by the parser.
	The string literal value is restricted in the same way as described in "*Text Number Character Restrictions*" in the description of the dfdl:textStandardDecimalSeparator

Comment [SMH322]: Errata 2.26 **DONE**

Comment [m323]: Revised Errata 2.26 Done

Comment [SMH324]: Text Number Character Restrictions
DONE

Comment [m325]: Errata 2.18 Done

Comment [SMH326]: Errata 2.18 clarifies allowable string literal content
DONE

Comment [SMH327]: Text Number Character Restrictions
DONE

Comment [m328]: Errata 2.18 Done

Comment [SMH329]: Errata 2.47 changes name **DONE**

Comment [SMH330]: Errata 2.18 clarifies allowable string literal content
DONE

Comment [SMH331]: Text Number Character Restrictions
DONE

Comment [m332]: Errata 2.18 Done

	property. Annotation: dfdl:element, dfdl:simpleType
textStandardZeroRep	List of DFDL String Literals Valid values: empty string, any character string The whitespace separated list of alternative literal strings that are equivalent to zero, for example the characters 'zero'. The representation is examined for a match to one of the values of this property after padding has been trimmed away. On unparsing the first value is used. If dfdl:ignoreCase is 'yes' then the case of the string is ignored by the parser. The empty string means that there is no special literal string for zero. This property is applicable when dfdl:textNumberRep is 'standard' and dfdl:textStandardBase is 10. The string literal is restricted to allow only certain kinds of syntax: • 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. It is a schema definition error if the string literal contains any of the disallowed syntax. Annotation: dfdl:element, dfdl:simpleType
textStandardBase	Non-negative Integer Valid Values 2, 8, 10, 16 Indicates the number base. Only used when dfdl:textNumberRep is 'standard'. When base is not 10, xs:decimal, xs:float and xs:double are not supported. When dfdl:textNumberRep is 'zoned' dfdl:textNumberBase is not used and base 10 is assumed. Annotation: dfdl:element, dfdl:simpleType
textZonedSignStyle	Enum Specifies the characters that are used to overpunch the sign nibble when the encoding is an ASCII-derived character set.encoding. The location of this sign nibble is indicated in the dfdl:textNumberPattern. This property is applicable when dfdl:textNumberRep is 'zoned'

Comment [SMH333]: Errata 2.42 clarifies allowable string literal content
DONE

Comment [SMH334]: Errata 2.25 adds rule about trimming

Comment [m335]: I thought about this and decided that detail doesn't belong here. It is discussed under textStandardPadCharacter.
DONE

Comment [SMH336]: Delete heading as never referenced (the rules for zero rep are unique).
DONE

Comment [m337]: Errata 2.42 Done

Comment [m338]: Since this is now a list of possibilities, it seems inconsistent not to allow ES. Why can't I have 'zero' or ES as the alternative zero indiations?

SMH: Fair point, suggest you table this as an item for discussion on WG call.

DONE (decided not to allow %ES because it adds some complexity to the empty representation processing rules. Can always make an element require and use default of 0.)

Comment [SMH339]: Errata 2.15. DONE

	Used only when dfdl:encoding specifies an ASCII-derived character set encoding. That is when the character code points 0x20 - 0x7E are assigned the same characters as in the US-ASCII encoding. This includes all the Unicode character set encodings, and all variations of ASCII and ISO-8859.
	Valid values 'asciiStandard', 'asciiTranslatedEBCDIC', 'asciiCARealiaModified', and 'asciiTandemModified'
	Which characters are used to represent 'overpunched' (included) positive and negative signs, varies by encoding, Cobol compiler and system. It is fixed for EBCDIC systems but not for ASCII.
	In EBCDIC-based encodings, characters '{ABCDEFGHI' or '0123456789' represent a positive sign and digits 0 to 9. (Character codes 0xC0 to 0xC9 or 0xF0 to 0xF9). The characters '}JKLMNOPQR' or '^£¥·©§¶¼½¾ 'represent a negative sign and digits 0 to 9. (character codes 0xD0 to 0xD9 or 0xB0 to 0xB9) On parsing both ranges of characters will be accepted. On unparsing the range 0xC0 to 0xC9 will be produced for positive signs and 0xD0 to 0xD9 will be produced for negative signs.
	asciiStandard: ASCII characters '0123456789' represent a positive sign and the corresponding digit. (Sign nibble for '+' is 0x3, which is the high nibble of these character codes unmodified.) ASCII characters 'pqrstuvwxy' represent negative sign and digits 0 to 9. (Character codes 0x70 to 0x79)
	asciiTranslatedEBCDIC: The overpunched character is the ASCII equivalent of the EBCDIC above. So the characters '{ABCDEFGHI' still represent a positive sign and digits 0 to 9. (These are character codes 0x7B, 0x41 through 0x49). The characters '}JKLMNOPQR' still represent negative sign and digits 0 to 9. (These are character codes 0x7D, 0x4A through 0x52). This case comes up if ebcdic zoned decimal data is translated to ASCII as if it were textual data.
	asciiCARealiaModified[23]: In this style, the ASCII characters '0123456789' represent positive sign and digits 0 to 9 as in standard. However, ASCII characters from code 0x20 to 0x29 are used for negative sign and the corresponding decimal

---

[23] Reference for this CA Realia 0x20 overpunch for negative sign is the article: "EBCDIC to ASCII Conversion of Signed Fields" at http://www.discinterchange.com/TechTalk_signed_fields_.html, where it says:

COBOL compilers that run on ASCII platforms have a "signed" data type that operates in a similar manner to the EBCDIC Signed field -- that is, they over punch the sign on the LSD. However, this is not standardized in ASCII, and different compilers use different overpunch codes. For example, Computer Associates' Realia compiler uses a 30 hex for positive values and a 20 hex for negative values, but Micro Focus® and Microsoft® use 30 hex for positive values and 70 hex for negative values.

	digit. This doesn't translate well into printing characters. These characters include the space (' ') for zero, characters '!"#$%&' for 1 through 6, the single quote character '"' for 7, and the parenthesis '()' for 8 and 9.
	asciiTandemModified: In this style the ascii characters '0-9' represent positive sign and digits 0 to 9, but bytes from 0x80 to 0x89 are used to represent overpunched negative sign and a digit. There are no corresponding character codepoints in the standard ASCII encoding since these values are all above 128 (decimal). (Note that neither ISO-8859-1 encoding nor Unicode have assigned glyphs for these codepoints. They are considered control characters.)
	Annotation: dfdl:element, dfdl:simpleType

Comment [m340]: Errata 2.88 Done

The dfdl:textStandardDecimalSeparator, dfdl:textStandardGroupingSeparator, dfdl:textStandardExponentRep, dfdl:textStandardInfinityRep, dfdl:textStandardNaNRep, and dfdl:textStandardZeroRep must all be distinct, and it is a schema definition error otherwise. Note that if dfdl:textStandardDecimalSeparator, dfdl:textStandardGroupingSeparator, or dfdl:textStandardExponentRep are expressions, this checking can only be carried out during processing (parsing or unparsing.)

Comment [m341]: Errata 2.106 *Section 13.6.* DONE

Implementation note: This rule is in the interests of clarity, and is an extra constraint compared to ICU.

13.6.1   The textNumberPattern Property

The dfdl:textNumberPattern describes how to parse and unparse text representations of number logical types with base 10.

The length of the representation of the number is determined first, and the number pattern is used only for conversion of the content text to and from a numeric logical infoset value.

The pattern described below is derived from the ICU DecimalFormat class described here: [ICUDecForm]

The pattern is an ICU-like syntax that defines where grouping separators, decimal separators, implied decimal points, exponents, positive signs and negative signs appear. It permits definition by either digits/fractions or significant digits.

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

13.6.1.1   dfdl:textNumberPattern for dfdl:textNumberRep 'standard'

When dfdl:textNumberRep is 'standard' this property only applies when dfdl:textStandardBase is 10

The pattern comes in two parts separated by a semi-colon. The first is mandatory and applies to positive numbers, the second is optional and applies to negative numbers.

Examples: The first shows digits/fractions and positive/negative signs, the second shows exponent, the third shows virtual decimal point, the fourth shows scaling position.

+###,##0.00;(###,##0.00)

##0.0#E0

000V00

PPP0000

The 'V' symbol is used to indicate the location of an implied decimal point for fixed point number representations. (This is an extension to the ICU pattern language.)

The 'P' symbol is used to indicate that a decimal scaling factor needs to be applied. (This is an extension to the ICU pattern language.)

The actual grouping separator, decimal separator and exponent characters are defined independently of the pattern.

The actual positive sign and negative sign are defined within the pattern itself.

Many characters in a pattern are taken literally; they are matched during parsing and output unchanged during unparsing. Special characters, on the other hand, stand for other characters, strings, or classes of characters. For example, the '#' character is replaced by a digit.

To insert a special character in a pattern as a literal, that is, without any special meaning, the character must be quoted. There are some exceptions to this which are noted below.

Symbol	Location	Meaning
0	Number	Digit
1-9	Number	'1' through '9' indicates rounding.
#	Number	Digit, zero shows as absent
.	Number	Decimal separator or monetary decimal separator
-	Number	Minus sign
,	Number	Grouping separator
E	Number	Separates mantissa and exponent in scientific notation. Need not be quoted in prefix or suffix.
+	Exponent	Prefix positive exponents with plus sign. Need not be quoted in prefix or suffix.
;	Subpattern boundary	Separates positive and negative subpatterns
'	Prefix or suffix	Used to quote special characters in a prefix or suffix, for example, "'#'#" formats 123 to "#123". To create a single quote itself, use two in a row: "# o''clock".
*	Prefix or suffix boundary	Pad escape, precedes pad character
V	Number	Virtual decimal point marker. Only used with decimal, float and double simple types.
P	Number	Decimal scaling position. Only used with decimal, float and double simple types.
@	Number	Significant digits specifier. Only used with decimal simple type. Controls number of significant digits when used alone or in conjunction with the # character.

Table 19 dfdl:textNumberPattern special characters

A pattern contains a positive and negative subpattern, for example, "#,##0.00;(#,##0.00)". Each subpattern has a prefix, a numeric part, and a suffix. If there is no explicit negative subpattern, the negative subpattern is the minus sign prefixed to the positive subpattern. That is, "0.00" alone is

Comment [SMH342]: Errata 2.28 adds @ symbol
DONE

Comment [SMH343]: Errata 2.40: Ensure the Location values match the BNF on the next page

Comment [m344]: (OPEN) ICU related. Please review carefully.

Comment [SMH345]: Errata in progress to remove float and double

DONE(no such errata)

Comment [m346]: DONE

SMH: Can't find such an errata? I think this errata may have ended up disallowing V and P in conjunction with E, rather than floats and doubles per se (errata 2.80).

equivalent to "0.00;-0.00". If there is an explicit negative subpattern, it serves only to specify the negative prefix and suffix; the number of digits, minimal digits, and other characteristics are ignored in the negative subpattern. That means that "#,##0.0#;(#)" has precisely the same result as "#,##0.0#;(#,##0.0#)".

The prefixes, suffixes, and various symbols used for infinity, digits, grouping separators, decimal separators, etc. may be set to arbitrary values, and they will appear properly during unparsing. However, care must be taken that the symbols and strings do not conflict, or parsing will be unreliable. For example, either the positive and negative prefixes or the suffixes must be distinct for parse to be able to distinguish positive from negative values.

The *grouping separator* is a character that separates clusters of integer digits to make large numbers more legible. It commonly used for thousands, but in some locales it separates ten-thousands. The *grouping size* is the number of digits between the grouping separators, such as 3 for "100,000,000" or 4 for "1 0000 0000". There are actually two different grouping sizes: One used for the least significant integer digits, the *primary grouping size*, and one used for all others, the *secondary grouping size.* In most locales these are the same, but sometimes they are different. For example, if the primary grouping interval is 3, and the secondary is 2, then this corresponds to the pattern "#,##,##0", and the number 123456789 is formatted as "12,34,56,789". If a pattern contains multiple grouping separators, the interval between the last one and the end of the integer defines the primary grouping size, and the interval between the last two defines the secondary grouping size. All others are ignored, so "#,##,###,####" == "###,###,####" == "##,#,###,####".

The P symbol is used to derive the location of an assumed decimal point when the point is not within the number that appears in the data. It acts as a decimal scaling factor.

The symbol P can be specified only as a continuous string of Ps in the leftmost or rightmost digit positions in the vpinteger region of the pattern.

It is a schema definition error if any symbols other than "0", "1" through "9" or # are used in the vpinteger region of the pattern.

Examples

Data representation	Pattern	Value
123	PP000	0.00123
123	000PP	12300

```
pattern     := subpattern (';' subpattern)?
subpattern := prefix? ((number exponent?)| vpinteger) suffix?
number      := (integer ('.' fraction)?) | sigdigits

vpinteger   := pinteger | (vinteger exponent?)
pinteger    := ('P'* integer) | (integer 'P'* )
vinteger    := ('V'? integer) |
               ('#'* 'V'? integer)|
               ('#'* '0'* 'V'? '0'* '0')|
               (integer 'V'?)

prefix      := '\u0000'..'\uFFFD' – specialCharacters
suffix      := '\u0000'..'\uFFFD' – specialCharacters
integer     := '#'* '0'* '0'
fraction    := '0'* '#'*
sigDigits   := '#'* '@' '@'* '#'*
exponent    := 'E'? '+'? '0'* '0'
padSpec     := '*' padChar
```

```
padChar     := '\u0000'..'\uFFFD' – quote

Notation:
  X*        0 or more instances of X
  X?        0 or 1 instances of X
  X|Y       either X or Y
  C..D      any character from C up to D, inclusive
  S-T       characters in S, except those in T
```

Figure 5 dfdl:textNumberPattern BNF syntax

The first subpattern is for positive numbers. The second (optional) subpattern is for negative numbers.

Not indicated in the BNF syntax above:

- The grouping separator ',' can occur inside the `integer` region, between any two pattern characters of that region, as long as the `number` region is not followed by an exponent region.

- Two grouping intervals are recognized: That between the decimal point and the first grouping symbol, and that between the first and second grouping symbols. These intervals are identical in most locales, but in some locales they differ. For example, the pattern "#,##,###" formats the number 123456789 as "12,34,56,789".

- The pad specifier `padSpec` may appear before the prefix, after the prefix, before the suffix, after the suffix, or not at all.

- In place of '0', the digits '1' through '9' in the `number` or `vpinteger` region may be used to indicate a rounding increment.

- The term *maximum fraction digits* is the total number of '0' and '#' characters in the `fraction` sub-pattern above.

- The term *minimum fraction digits* is the total number of '0' characters (only) in the `fraction` sub-pattern above.

- The term *maximum integer digits* is a limit that is implementation dependent, but must be at least 20 (which is the number of digits in a base 10 unsigned long).[24]

- The term *minimum integer digits* is the total number of '0' characters (only) in the `integer` sub-pattern above.

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

Parsing

During parsing, grouping separators are removed from the data.

---

[24] Implementations which use current versions of the popular ICU library will allow 309 digits as *maximum integer digits*.

Comment [SMH354]: Errata 2.28 adds significant digits so BNF must change too

DONE

Comment [m355]: (OPEN) ICU Related. Needs to be reviewed carefully.

Comment [SMH356]: Errata 2.28 adds significant digits
DONE.

Comment [SMH357]: Updated 2.29. DONE

Comment [m358]: 2.29 Updated - changed to implementation dependent, at least 20.

Comment [m359]: Errata 2.29 Done

Comment [m360]: Errata 2.79 DONE

Comment [m361]: Errata 2.80 Done

Comment [SMH362]: Errata 2.54 adds more parsing specific behavior. But is incomplete still.

DONE

Unparsing

Unparsing is guided by several parameters all of which can be specified using a pattern. The following description applies to formats that do not use scientific notation.

- If the number of actual integer digits exceeds the *maximum integer digits*, then only the least significant digits are shown. For example, 1997 is formatted as "97" if the maximum integer digits is 2.

- If the number of actual integer digits is less than the *minimum integer digits*, then leading zeros are added. For example, 1997 is formatted as "01997" if the minimum integer digits is 5.

- If the number of actual fraction digits exceeds the *maximum fraction digits*, then half-even rounding it performed to the maximum fraction digits. For example, 0.125 is formatted as "0.12" if the maximum fraction digits is 2. This behavior can be changed by specifying a rounding increment and a rounding mode.

- If the number of actual fraction digits is less than the *minimum fraction digits*, then trailing zeros are added. For example, 0.125 is formatted as "0.1250" if the minimum fraction digits is 4.

- Trailing fractional zeros are not displayed if they occur *j* positions after the decimal, where *j* is less than the maximum fraction digits. For example, 0.10004 is formatted as "0.1" if the maximum fraction digits is four or less.

Comment [SMH363]: Errata 2.29 requires this to be reworded

DONE

Special Values

`NaN` is represented as a string determined by the dfdl:textStandardNaNRep property. This is the only value for which the prefixes and suffixes are not used.

Comment [SMH364]: Errata 2.47 changes the name **DONE**

Infinity is represented as a string with the positive or negative prefixes and suffixes applied. The infinity string is determined by the dfdl:textStandardInfinityRep property.

Scientific Notation

Numbers in scientific notation are expressed as the product of a mantissa and a power of ten, for example, 1234 can be expressed as 1.234×10^3. The mantissa is typically in the half-open interval [1.0, 10.0) or sometimes [0.0, 1.0), but it need not be. In a pattern, the exponent character immediately followed by one or more digit characters indicates scientific notation. Example: "0.###E0" formats the number 1234 as "1.234E3".

- The number of digit characters after the exponent character gives the minimum exponent digit count. There is no maximum. Negative exponents are formatted using the minus sign, *not* the prefix and suffix from the pattern. This allows patterns such as "0.###E0 m/s". To prefix positive exponents with a plus sign, specify '+' between the exponent and the digits: "0.###E+0" will produce formats "1E+1", "1E+0", "1E-1", etc.

- The minimum number of integer digits is achieved by adjusting the exponent. Example: 0.00123 formatted with "00.###E0" yields "12.3E-4". This only happens if there is no maximum number of integer digits. If there is a maximum, then the minimum number of integer digits is fixed at one.

- The maximum number of integer digits, if present, specifies the exponent grouping. The most common use of this is to generate *engineering notation*, in which the exponent is a multiple of three, e.g., "##0.###E0". The number 12345 is formatted using "##0.####E0" as "12.345E3".

- When using scientific notation, the formatter controls the digit counts using significant digits logic. The maximum number of significant digits limits the total number of integer and fraction digits that will be shown in the mantissa; it does not affect parsing. For example, 12345 formatted with "##0.##E0" is "12.3E3". .

- Exponential patterns may not contain grouping separators.

Significant Digits

The '@' pattern character can be used with the '#' to control how many integer and fraction digits are needed to display the specified number of significant digits. The '@' only affects unparsing behavior. Examples:

Pattern	Minimum significant digits	Maximum significant digits	Number	Formatted Output
@@@	3	3	12345	12300
@@@	3	3	0.12345	0.123
@@##	2	4	3.14159	3.142
@@##	2	4	1.23004	1.23

- Significant digit counts may be expressed using patterns that specify a minimum and maximum number of significant digits. These are indicated by the '@' and '#' characters. The minimum number of significant digits is the number of '@' characters. The maximum number of significant digits is the number of '@' characters plus the number of '#' characters following on the right. For example, the pattern "@@@" indicates exactly 3 significant digits. The pattern "@##" indicates from 1 to 3 significant digits. Trailing zero digits to the right of the decimal separator are suppressed after the minimum number of significant digits have been shown. For example, the pattern "@##" formats the number 0.1203 as "0.12".

- If a pattern uses significant digits, it may not contain a decimal separator, nor the '0' pattern character. Patterns such as "@00" or "@.###" are disallowed.

- Any number of '#' characters may be prepended to the left of the leftmost '@' character. These have no effect on the minimum and maximum significant digits counts, but may be used to position grouping separators. For example, "#,#@#" indicates a minimum of one significant digits, a maximum of two significant digits, and a grouping size of three.

- The number of significant digits has no effect on parsing.

- Significant digits may be used together with exponential notation. For example, the pattern "@@###E0" is equivalent to "0.0###E0".

- The '@' pattern character can be used only in 'standard' textNumberRep (not 'zoned'), and excludes the 'P' and 'V' pattern characters. It is a schema definition error if the '@' pattern character appears in 'zoned' textNumberRep, or in conjunction with the 'P' or 'V' pattern characters.

Padding

Padding may be specified through the pattern syntax. In a pattern the pad escape character, followed by a single pad character, causes padding to be parsed and formatted. The pad escape character is '*'. For example, "*x#,##0.00" formats 123 to "xx123.00", and 1234 to "1,234.00".

- When padding is in effect, the width of the positive subpattern, including prefix and suffix, determines the format width. For example, in the pattern `"* #0 o''clock"`, the format width is 10.

- The width is counted in 16-bit code units.

- Some parameters which usually do not matter have meaning when padding is used, because the pattern width is significant with padding. In the pattern `"* ##,##,#,##0.##"`, the format width is 14. The initial characters `"##,##,"` do not affect the grouping size or maximum integer digits, but they do affect the format width.

- Padding may be inserted at one of four locations: before the prefix, after the prefix, before the suffix, or after the suffix. If there is no prefix, before the prefix and after the prefix are equivalent, likewise for the suffix.

- When specified in a pattern, the 32-bit codepoint immediately following the pad escape is the pad character. This may be any character, including a special pattern character. That is, the pad escape *escapes* the following character. If there is no character after the pad escape, then the pattern is illegal.

Note: Padding specified through the pattern syntax is distinct from, and in addition to, padding specified using dfdl:textPadKind.

Rounding

How rounding is controlled is given by dfdl:textNumberRounding. The rounding increment may be specified in the dfdl:textNumberPattern itself using digits '1' through '9' or using an explicit increment in dfdl:textNumberRoundingIncrement. For example, 1230 rounded to the nearest 50 is 1250. 1.234 rounded to the nearest 0.65 is 1.3.

- Rounding only affects the string produced by unparsing. It does not affect parsing or change any numerical values.

- In a pattern, digits '1' through '9' specify rounding, but otherwise behave identically to digit '0'. For example, `"#,#50"` specifies a rounding increment of 50.

- Using digits in a pattern, rounding is always 'half even', meaning rounds towards the nearest integer, or towards the nearest even integer if equidistant.

Using an explicit rounding increment, dfdl:textNumberRoundingMode determines how values are rounded.

13.6.1.2   dfdl:textNumberPattern for dfdl:textNumberRep 'zoned'

When dfdl:textNumberRep is 'zoned' a subset of the number pattern language described in Section 13.6.1.1 dfdl:textNumberPattern for dfdl:textNumberRep 'standard' is used.

Only the pattern for positive numbers is used. It is a schema definition error if the negative pattern is specified.

In addition, only the following pattern characters may be used:

- '+' MUST BE present at the beginning or end of the pattern to indicate whether the leading or trailing digit carries the overpunched sign, if the logical type is signed

- '+' MAY BE present at the beginning or end of the pattern to indicate whether the leading or trailing digit carries the overpunched sign, if the logical type is unsigned. If logical type is unsigned and dfdl:textNumberPolicy = 'lax' specified it is a schema definition error if no '+' is present.

Comment [m369]: DONE

What if you have both? SDE?

SMH: You can't – it's controlled by textNumberRounding property 'pattern' v 'explicit'. (Though there is more to discuss on this as per 18th Sept WG call.) DONE

Comment [SMH370]: Errata 2.41 implies rewrite of this section to relate it to 13.6.1.1. DONE

- 'V' MAY BE used to indicate the location of an implied decimal point

- 'P' MAY BE used to indicate the decimal scaling

- '0-9' indicates the number of needed digits (including overpunched).

- '#' indicates the number of optional digits.

Rounding occurs as described under Rounding in 13.6.1.1 dfdl:textNumberPattern for dfdl:textNumberRep 'standard'

13.6.2 Converting logical numbers to/from text representation

- Signed numbers with dfdl:textNumberRep 'standard' and dfdl:textStandardBase 10 are mapped using the dfdl:textNumberPattern.

- Signed numbers with dfdl:textNumberRep 'standard' and dfdl:textStandardBase not 10 are mapped to an unsigned representation. On unparsing the minimum number of characters to represent the digits is output and it is a processing error if the value is negative.

- Signed numbers with dfdl:textNumberRep 'zoned' are mapped using the dfdl:textNumberPattern to indicate the position of the sign and virtual decimal point. On parsing if the sign is not overpunched, that is it does not have a sign, it is treated as positive. On unparsing the sign is always overpunched.

- Unsigned numbers with dfdl:textNumberRep 'standard' and dfdl:textStandardBase 10 are mapped using the dfdl:textNumberPattern. On parsing it is a processing error if the data are negative.

- Unsigned numbers with dfdl:textNumberRep 'standard' and dfdl:textStandardBase not 10 are mapped to an unsigned representation. On unparsing the minimum number of characters to represent the digits is output. .

- Unsigned numbers with dfdl:textNumberRep 'zoned' are mapped using the dfdl:textNumberPattern to indicate the position of the sign and virtual decimal point. On parsing it is a processing error if the data are negative. On unparsing the data are not overpunched with a sign.

Comment [SMH371]: 'of'

Comment [m372]: Errata 2.28 also applies here I expect. ('@' pattern feature)

Resolved: leave out for now. May need to be added back in, but let's wait and see.

Comment [m373]: DONE

Comment [SMH374]: Consider adding BNF syntax

Comment [m375]: (OPEN)

Comment [SMH376]: Font size

13.7   Properties Specific to Numbers with Binary representation

These properties are applicable to decimals types and its derived types. These properties are not applicable to types float and double. See section 1.1

Comment [m377]: **Errata 2.99**. *Section 13.7, 13.13*. DONE

.

Property Name	Description
binaryNumberRep	Enum Valid values are 'packed', 'bcd', 'binary', 'ibm4690Packed' Allowable values for each number type are: {table below} 'packed' means represented as an IBM 390 packed decimal. Each byte contains two decimal digits, except for the least significant byte, which contains a sign in the least significant nibble. 'bcd' means represented as a binary coded decimal with two digits per byte. 'binary' means represented as twos complement for signed types and unsigned binary for unsigned types. Note that the maximum allowed value for twos-complement binary integers is implementation dependent, but must be at least that of a xs:long type, which is the equivalent of an 8 byte/64-bit signed integer. 'ibm4690Packed' is a variant of a packed decimal having 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. For all values, the dfdl:byteOrder property is used to determine the numeric significance of the bytes making up the representation. Annotation: dfdl:element, dfdl:simpleType

Logical Type	Permitted Value
Decimal, Integer, NonNegativeInteger	packed, bcd, binary, ibm4690Packed
Long, Int, Short, Byte,	packed, binary, ibm4690Packed (but not bcd)
UnsignedLong, UnsignedInt, UnsignedShort, UnsignedByte	packed, bcd, binary, ibm4690Packed

Comment [m378]: Should binary (twos-complement) be allowed for these types?

(DONE)

SMH: decided to allow this
DONE

Comment [m379]: Errata **2.98**. *Section 13.7*. DONE

Comment [SMH380]: Errata 2.38: Add sensitivity to byteOrder

DONE

binaryDecimalVirtualPoint	Integer.
	Used when base simpleType is xs:decimal.
	An integer that represents the position of an implied decimal point within a number, or specify 0.
	If you specify 0 then there is no virtual decimal point
	If you specify a positive integer, the position of the decimal point is moved left from the right side of the number. For example, if you specify 3, the decimal value 1234 represents 1.234
	If you specify a negative integer, the position of the decimal point is moved right from the right side of the number. For example, if you specify -3, the decimal value 1234 represents 1 234 000
	Annotation: dfdl:element, dfdl:simpleType
binaryPackedSignCodes	List of Character
	Used only when dfdl:binaryNumberRep or dfdl:binaryCalendarRep is 'packed'
	A space separated string giving the hex sign nibbles to use for a positive value, a negative value, an unsigned value, and zero.
	Valid values for positive nibble: A, C, E, F
	Valid values for negative nibble: B, D
	Valid values for unsigned nibble: F
	Valid values for zero sign: A C E F 0
	Example: 'C D F C' – typical S/390 usage
	Example: 'C D F 0' – handle special case for zero
	On parsing, whether to accept all valid values for a positive, negative or unsigned number, and for zero, is governed by the dfdl:binaryNumberCheckPolicy property. On unparsing, the specified values are always used.
	Annotation: dfdl:element, dfdl:simpleType
binaryNumberCheckPolicy	Enum
	Values are 'strict' and 'lax'.
	Indicates how lenient to be when parsing binary numbers.
	If 'lax' then the parser tolerates all valid alternatives where such alternatives exist. Specifically, for dfdl:binaryNumberRep = 'packed' the sign nibble for positive, negative, unsigned and zero is allowed to be any of the valid respective values.
	On unparsing, the specified value is always used
	Annotation: dfdl:element, dfdl:simpleType

13.7.1 Converting Logical Numbers to/from Binary Representation

Signed numbers with dfdl:binaryNumberRep 'packed' are parsed using sign nibble to indicate the sign. The unsigned nibble is treated as positive. On unparsing the sign nibble is written according to dfdl:binaryPackedSignCodes. The unsigned nibble is never written.

Signed numbers with dfdl:binaryNumberRep 'bcd' are always positive. On unparsing it is a processing error if the data is negative.

Signed numbers with dfdl:binaryNumberRep 'ibm4690Packed' are parsed using the sign nibble to identify negative values. There is no sign nibble for positive values. On unparsing the nibble 0xD is written for negative values.

Comment [m381]: Errata 2.99 DONE

Unsigned numbers with dfdl:binaryNumberRep 'packed' are parsed if the nibble is positive or unsigned. It is a processing error if the data is negative. On unparsing the unsigned nibble is used.

Unsigned numbers with dfdl:binaryNumberRep 'bcd' are readily parsed as BCD data is always positive.

Unsigned numbers with dfdl:binaryNumberRep 'ibm4690Packed' are parsed if there is no sign nibble of 0xD to identify a negative value. It is a processing error if the data is negative. On unparsing no sign nibble is written.

Comment [m382]: Errata 2.99 DONE

When unparsing a number and excess precision is supplied in the Infoset no rounding occurs. It is a processing error.

Comment [m383]: Errata 2.93. Sections 13.6, 13.7. DONE.

13.8 Properties Specific to Float/Double with Binary representation

Property Name	Description
binaryFloatRep	Enum or DFDL Expression This specifies the encoding method for the float and double. Valid values are 'ieee', 'ibm390Hex'. This property can be computed by way of an expression which returns a string of 'ieee' or ' ibm390Hex' . The expression must not contain forward references to elements which have not yet been processed. The enumeration value 'ieee' refers to the IEEE 754-1985 specification. For both 'ieee' and 'ibm390hex', an xs:float must have a physical length of 4 bytes. It is a schema definition error if there is a specified length not equivalent to 4 bytes. Similarly, for both 'ieee' and 'ibm390hex', an xs:double must have a physical length of 8 bytes. It is a schema definition error if there is a specified length not equivalent to 8 bytes. The byteOrder property is used to construct a value from the bytes in the binary representation. Keep in mind that the DFDL Infoset float and double data types match the precision of the IEEE specification. There may be precision/rounding issues when converting IBM float/double to/from the DFDL infoset float/double types. Half-precision IEEE and quad-precision IEEE/IBM are not supported.[25] Annotation: dfdl:element, dfdl:simpleType

Comment [SMH384]: Errata 3.5 clarifies exactly which IEEE variant is supported

DONE

Comment [SMH385]: Errata 3.5 clarifies possible rounding/precision issues with IBM floats
DONE

Comment [SMH386]: Errata 3.5 places restrictions on the physical lengths

DONE

Comment [SMH387]: Errata 2.38: Add sensitivity to byteOrder

DONE

13.9 Properties Specific to Boolean with Text representation

Property Name	Description
textBooleanTrueRep	List of DFDL String Literals or DFDL Expression A whitespace separated list of representations to be used for 'true'. These are compared after trimming when parsing, and before padding when unparsing. If lengthKind is 'explicit' or 'implicit' and either textPadKind or textTrimKind is 'none' then both textBooleanTrueRep and textBooleanFalseRep must have the same length else it is a

Comment [SMH388]: Errata 2.19 clarifies allowable string literal content
DONE

Comment [SMH389]: Errata 2.30 adds rules for padding, trimming and length
DONE

---

[25] Note 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.

	schema definition error.
	This property can be computed by way of an expression which returns a string of whitespace separated list of values. The expression must not contain forward references to elements which have not yet been processed.
	On unparsing the first value is used
	If dfdl:ignoreCase is 'yes' then the case of the string is ignored by the parser.
	Text Boolean Character Restrictions: The string literal is restricted to allow only certain kinds of syntax:
	• DFDL character entities are allowed
	• The raw byte entity (%#r) is not allowed.
	• DFDL Character classes (NL, WSP, WSP+, WSP*, ES) are not allowed
	It is a schema definition error if the string literal contains any of the disallowed constructs.
	Annotation: dfdl:element, dfdl:simpleType
textBooleanFalseRep	List of DFDL String Literals or DFDL Expression
	A whitespace separated list of representations to be used for 'false'. These are compared after trimming when parsing, and before padding when unparsing.
	If lengthKind is 'explicit' or 'implicit' and either textPadKind or textTrimKind is 'none' then both textBooleanTrueRep and textBooleanFalseRep must have the same length else it is a schema definition error.
	This property can be computed by way of an expression which returns a string of whitespace separated list of values. The expression must not contain forward references to elements which have not yet been processed.
	On unparsing the first value is used
	If dfdl:ignoreCase is 'yes' then the case of the string is ignored by the parser.
	The string literal value is restricted in the same way as described in "*Text Boolean Character Restrictions*" in the description of the textBooleanTrueRep property.
	Annotation: dfdl:element, dfdl:simpleType
textBooleanJustification	Enum
	Valid values 'left', 'right', 'center'
	Controls how the data is padded or trimmed on parsing and unparsing.
	Behavior as for dfdl:textStringJustification.
	Annotation: dfdl:element, dfdl:simpleType
textBooleanPadCharacter	DFDL String Literal

Comment [SMH390]: Text Boolean Character Restrictions DONE

Comment [m391]: Errata 2.19 Done

Comment [SMH392]: Errata 2.19 clarifies allowable string literal content DONE

Comment [SMH393]: Errata 2.30 adds rules for padding, trimming and length DONE

Comment [SMH394]: Text Boolean Character Restrictions DONE

Comment [SMH395]: Errata 2.17 clarifies allowable string literal content DONE

	The value that is used when padding or trimming boolean elements. The value can be a single character or a single byte.
	If a character, then it can be specified using a literal character or using DFDL entities.
	If a byte, then it must be specified using a single byte value entity
	If a pad character is specified when lengthUnits='bytes' then the pad character must be a single-byte character.
	If a pad byte is specified when lengthUnits='characters' then
	• the encoding must be a fixed-width encoding
	• padding and trimming must be applied using a sequence of N pad bytes, where N is the width of a character in the fixed-width encoding.
	The string literal value is restricted in the same way as described in "Padding Character Restrictions" in the description of the textStringPadCharacter property.
	Annotation: dfdl:element, dfdl:simpleType

Comment [SMH396]: Padding Character Restrictions
DONE

Comment [m397]: Errata 2.17 Done

13.10 Properties Specific to Boolean with Binary Representation

Property Name	Description
binaryBooleanTrueRep	Non-negative Integer
	Representation to be used for 'true'
	The empty string means dfdl:binaryBooleanTrueRep is any value other than dfdl:binaryBooleanFalseRep.
	If provided, then if the element has specified length, then is is a schema definition error if the length is not between 1 bit and 32 bits (4 bytes). It is also a schema definition error if the value of the binaryBooleanTrueRep cannot fit as an unsigned binary integer in the specified length.
	Annotation: dfdl:element, dfdl:simpleType
binaryBooleanFalseRep	Non-negative Integer
	Representation to be used for 'false'
	If the element has specified length, then is is a schema definition error if the length is not between 1 bit and 32 bits (4 bytes). It is also a schema definition error if the value of the binaryBooleanFalseRep cannot fit as an unsigned binary integer in the specified length.
	Annotation: dfdl:element, dfdl:simpleType

13.11 Properties specific to Calendar with Text or Binary representation

The properties describe how a calendar is to be interpreted including a unparsing pattern property plus properties that qualify the pattern.

These properties can be used when a calendar has dfdl:representation 'text' or dfdl:representation 'binary' and a packed decimal representation.

Property Name	Description
calendarPattern	String Defines the ICU pattern that describes the format of the calendar. The pattern defines where the year, month, day, hour, minute, second, fractional second and time zone components appear. See calendarPattern property section below. When the dfdl:representation is 'binary' and the representation is a packed decimal then the pattern can contain only characters and symbols that always result in the presentation of digits. Annotation: dfdl:element, dfdl:simpleType
calendarPatternKind	Enum Valid values 'explicit', 'implicit' 'explicit' means the pattern is given by dfdl:calendarPattern, 'implicit' means the pattern is derived from the XML schema date/time type. {table below} Annotation: dfdl:element, dfdl:simpleType
calendarCheckPolicy	Enum Valid values are 'strict', 'lax' Indicates how lenient to be when parsing against the pattern. If 'lax' then the parser will convert invalid date/time values to the appropriate in-band value. For example, a date of 2005-05-32 will be converted to 2005-06-01. Annotation: dfdl:element, dfdl:simpleType
calendarTimeZone	String This property provides the time zone that will be assumed if no time zone explicitly occurs in the data. Valid values specify a UTC time zone offset by matching the regular expression: `(UTC)([+\-]([01]\d|\d)((([:][0-5]\d){1,2})?))?` In addition, empty string can be specified to indicate "no time zone", or the IANA time zone format (also known as the Olson time zone format) may be used. (e.g, America/New_York)) See [OLSON]. Note that this property is used when parsing only. Annotation: dfdl:element, dfdl:simpleType
calendarObserveDST	Enum Valid values are 'yes', 'no'

Inner table for calendarPatternKind:

Logical Type	Default Pattern
xs:date	yyyy-MM-dd
xs:dateTime	yyyy-MM-dd'T'HH:mm:ss
xs:time	HH:mm:ssZZZ

Comment [SMH398]: Errata 2.39. State what happens when parts are absent from pattern.

DONE (below in description section)

Comment [SMH399]: Errata 2.66: State what happens when parts in the patern but absent rom the data (incl timezone)

DONE (below in description section)

Comment [m400]: OPEN
And what does 'strict' mean??

Comment [SMH401]: Errata 2.43 changes this to a constrained string

DONE

Comment [SMH402]: Redundant
DONE

Comment [SMH403]: Errata 2.51: Applies to parsing only
DONE

Comment [SMH404]: Errata 2.50 empty string is allowed.

DONE

Comment [SMH405]: Errata 2.65 allows Olson format

DONE

	Whether the time zone given in dfdl:calendarTimeZone observes daylight savings time.	
	Ignored if calendarTimeZone is specified in UTC format, or if calendarTimeZone is empty string. That is, this property is used only if the calendarTimeZone is in IANA (aka Olson) format.	
	This property applies to parsing only.	
	Annotation: dfdl:element, dfdl:simpleType	
calendarFirstDayOfWeek	Enum	
	Valid values 'Monday' … 'Sunday'	
	The day of the week upon which a new week is considered to start.	
	Annotation: dfdl:element, dfdl:simpleType	
calendarDaysInFirstWeek	Non-negative Integer	
	Valid values 1 to 7	
	Specify the number of days of the new year that must fall within the first week.	
	The start of a year usually falls in the middle of a week. If the number of days in that week is less than the value specified here, the week is considered to be the last week of the previous year; hence week 1 starts some days into the new year. Otherwise it is considered to be the first week of the new year; hence week 1 starts some days before the new year.	
	Annotation: dfdl:element, dfdl:simpleType	
calendarCenturyStart	Non-negative Integer	
	Valid values 0 to 99.	
	This property determines on parsing how two-digit years are interpreted. Specify the two digits that start a 100-year window that contains the current year. For example, if you specify 89, and the current year is 2006, all two-digit dates are interpreted as being in the range 1989 to 2088. A two-digit year less than 89 will be interpreted as 20nn and a two-digit year more than or equal to 89 will be treated as 19nn.	
	Annotation: dfdl:element, dfdl:simpleType	
calendarLanguage	String	
	The language that is used when the pattern produces a presentation in text. For example 'Monday'	
	The valid values must match the regular expression:	
	`(([A-Za-z]{1,8}([\-][A-Za-z0-9]{1,8})*)`	
	Annotation: dfdl:element, dfdl:simpleType	

Comment [SMH406]: Errata 2.51: Applies to parsing only

DONE

Comment [SMH407]: Errata 2.65: Applies only when calendarTimeZone is Olson format

DONE

Comment [SMH408]: Wrong. This property is ignored if dfdl:calendarTimeZone is UTC format or is empty string. It only works when OLSON format, as it needs OLSON precision to know when to apply the DST difference.

FIXED. DONE

Comment [SMH409]: Errata 2.44 changes this to be a constrained string

DONE

Comment [SMH410]: Redundant DONE

Comment [SMH411]: Remove underscoring DONE

13.11.1 The dfdl:calendarPattern property

The dfdl:calendarPattern describes how to parse and unparse text and binary representations of dateTime, date and time logical types. The pattern is primarily used on unparsing to define the format but is also used to aid parsing.

The pattern is derived from the ICU SimpleDatetimeFormat class described here: [ICUCalForm]

An extension is the formatting symbols I which means accept a subset of ISO 8601 compliant calendars

Another extension is the use of 'U', to provide modifications of formatting symbols IU and ZU which specifies special use of 'Z' to represent the UTC time zone.

Symbol	Meaning	Presentation	Example	
G	era designator	Text	G	AD
y	year	Number	y, yyyy	1996
			yy	96
u	year (allows negative years)	Number	u	1900, 0, -500
Y	year (of the week of year)	Number	Y	1997
M	month in year	Text & Number	M, MM	09
			MMM	Sept
			MMMM	September
			MMMMM	S
d	day in month	Number	d	2
			dd	02
h	hour in am/pm (1~12)	Number	h	7
			hh	07
H	hour in day (0~23)	Number	H	0
			HH	00
m	minute in hour	Number	m	4
			mm	04
s	second in minute	Number	s	5
			ss	05
S	fractional second (see note 1)	Number	S	2
			SS	24
			SSS	235
E	day of week	Text	E, EE, EEE	Tues
			EEEE	Tuesday
			EEEEE	T
e	day of week (local)	Text & Number	e, ee	2
			eee	Tues
			eeee	Tuesday
			eeeee	T
D	day in year	Number	D	189
F	day of week in month	Number	F	2 (2nd Wed in July)

Comment [m412]: Errata 2.3. Done

Comment [SMH413]: Errata 2.68: Deleted T symbol. **DONE**

Comment [SMH414]: Errata 2.52 adds 'u' symbol
DONE

Comment [SMH415]: I would cover the fact the zz & zzz behave like z in a note below the table. (Same for ZZ & ZZZ) (DONE)

Comment [SMH416]: Need to say whether (Text) or (Number)
DONE

Comment [SMH417]: Need to say (Text)
DONE

Comment [SMH418]: Errata 2.3 adds more to this

Comment [m419]: Done

Comment [SMH420]: Missing 'I'
DONE

Comment [SMH421]: Delete empty row to keep table on one page ?
DONE

Comment [SMH422]: Errata 2.68: Deleted T symbol. **DONE**

w	week in year	Number	w, ww	27
W	week in month	Number	W	2
a	am/pm marker	Text	A	pm
k	hour in day (0~24)	Number	k	2, 24
			kk	02, 24
K	hour in am/pm (0~11)	Number	K	0
			KK	00
z	Time Zone: specific non-location	Text	z, zz, zzz	PDT
			zzzz	Pacific Daylight Time
Z	Time Zone: RFC 822	Text	Z, ZZ, ZZZ	-0800, +0000
	Time Zone: localized GMT		ZZZZ	GMT-08:00, GMT+00:00
ZU	As Z but with output "Z" if the time zone is +00:00	Text	ZU, ZZU, ZZZU	-0800, Z
			ZZZZU	GMT-08:00, Z
v	Time Zone: generic non-location	Text	v	PT
			vvvv	Pacific Time
V	Time Zone: generic non-location	Text	V	PT
			VVVV	United States (Los Angeles)
I	ISO8601 Date/Time	Text	I	2006-10-07T12:06:56.568+01:00
IU	As I but with output "Z" if the time zone is +00:00	Text	IU	2006-10-07T12:06:56.568Z
'	escape for text	Delimiter	'	'Date='
"	single quote	Literal	"	'o''clock'

> **Comment [m423]:** DONE
> Needs careful review - since the whole table was recast. (E.g., are these all (Text)?? Is ZU and ZZZZU supposed to have the exact same description??
>
> Reviewed and updated by Steve
>
> DONE

Any number of fractional seconds "S" may by specified in the pattern and accepted by implementations, but an implementation is free to represent a limited number of fractional seconds internally. Round up rounding must occur when converting between external and internal representations. At least millisecond accuracy must be implemented. Unlike other fields, fractional seconds are padded on the right with zero.

The count of pattern letters determines the format as indicated in the table.

> **Comment [SMH424]:** Expanded table to show the effect of the count of pattern letters, to match the ICU web pages. Text paragraphs deleted as a result.
>
> DONE

If dfdl:representation is text, any characters in the pattern that are not in the ranges of ['a'..'z'] and ['A'..'Z'] will be treated as quoted text. For instance, characters like ':', '.', ' ', '#' and '@' will appear in the formatted output even if they are not embraced within single quotes. The single quote is used to 'escape' letters. If dfdl:representation is binary, any characters in the pattern that are not digits must be quoted.

Two single quotes in a row, whether inside or outside a quoted sequence, represent a 'real' single quote.

> **Comment [SMH425]:** Clarified
>
> DONE

The symbols 'z', 'zz', and 'zzz' have identical meaning, as do Z', 'ZZ', and 'ZZZ', and 'ZU', 'ZZU', and 'ZZZU'.

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.

When parsing, for any pattern that omits components the values for the omitted components are supplied from the Unix epoch 1970-01-01T00:00:00.000.[27]

When unparsing, and the pattern contains a formatting symbol that requires a component of the date/time and the infoset value does not contain that component, it is a processing error.

When parsing a calendar element with a packed decimal representation 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 pattern according to the usual rules.

13.12 Properties specific to calendar with Text representation

Property Name	Description
textCalendarJustification	Enum Valid values 'left', 'right', 'center' Controls how the data is padded or trimmed on parsing and unparsing. Behavior as for dfdl:textStringJustification. Annotation: dfdl:element, dfdl:simpleType
textCalendarPadCharacter	DFDL String Literal The value that is used when padding or trimming calendar elements. The value can be a single character or a single byte. If a character, then it can be specified using a literal character or using DFDL entities. If a byte, then it must be specified using a single byte value entity If a pad character is specified when dfdl:lengthUnits='bytes' then the pad character must be a single-byte character.

---

[27] Note that DFDL does not support a pure month or day or year, as it does not support the XSD simple types xs:gMonth, xs:gDay, and xs:gYear.

Comment [SMH426]: Strictly "For any pattern (or 'I' symbol granularity)…"

Comment [m427]: Nah. I think that makes it less clear actually. – MikeB
DONE
SMH: OK
DONE

Comment [SMH428]: This is a specific instance of a more general rule, namely that when parsing/unparsing, if the pattern contains a component and that component is not present in the data/Infoset, it is a processing error.
DONE

Comment [m429]: Errata 2.104. *Section 13.11.1.* DONE

Comment [m430]: Errata 2.39 DONE

Comment [m431]: Done

Comment [SMH432]: Errata 2.17 clarifies allowable string literal content
DONE

	If a pad byte is specified when dfdl:lengthUnits='characters' then - the encoding must be a fixed-width encoding - padding and trimming must be applied using a sequence of N pad bytes, where N is the width of a character in the fixed-width encoding. The string literal value is restricted in the same way as described in *"Padding Character Restrictions"* in the description of the textStringPadCharacter property. Annotation: dfdl:element, dfdl:simpleType

Comment [SMH433]: Padding Character Restrictions
DONE

Comment [m434]: Errata 2.17 Done

13.13 Properties specific to Calendar with Binary Representation

Property Name	Description

binaryCalendarRep	Enum
	
	Valid values are 'packed', 'bcd', 'ibm4690Packed', 'binarySeconds', 'binaryMilliseconds'
	
	For all values, the dfdl:byteOrder property is used to determine the numeric significance of the bytes making up the representation.
	
	'packed' means represented as an IBM 390 packed decimal. Each byte contains two decimal digits, except for the rightmost byte, which contains a sign to the right of a decimal digit.
	
	'bcd' means represented as a binary coded decimal with two digits per byte.
	
	'ibm4690Packed' means represented as a variant of packed format as described in property dfdl:binaryNumberRep.
	
	For packed decimals the following properties are also applicable
	    • dfdl:binaryPackedSignCodes ('packed' only)
	    • dfdl:binaryNumberCheckPolicy
	
	For packed decimals the property dfdl:calendarPatternKind must be 'explicit' because the default patterns for 'implicit' use non-numeric characters. It is a schema definition error otherwise. Similarly, dfdl:calendarPattern can contain only characters and symbols that always result in the presentation of digits.
	
	See Properties specific to numbers with binary representation section : 13.7 Properties Specific to Numbers with Binary representation.
	
	Note also that a virtual decimal point for the boundary between seconds and fractional seconds is implied from the pattern at the boundary of 's' and 'S', i.e., where the substring 'sS' appears in the pattern.
	
	'binarySeconds' means represented as a 4 byte signed integer that is the number of seconds from the epoch (positive or negative). It is a schema definition error if there is a specified length not equivalent to 4 bytes.
	
	'binaryMilliseconds' means represented as an 8 byte signed

Comment [m435]: Errata 2.99. *Section 13.7, 13.13*. DONE

Comment [SMH436]: Errata 2.38: Add sensitivity to byteOrder

DONE

Comment [SMH437]: Reads better: "…must be 'explicit', because…"

DONE

Comment [SMH438]: "default patterns" DONE

Comment [SMH439]: Just say "It is a schema definition error otherwise" DONE

Comment [SMH440]: Errata 2.2 clarifies and changes this.

Comment [m441]: Done

Comment [SMH442]: I think the VDP words read better if placed AFTER the ref to 13.7.

Fixed. DONE

Comment [m443]: Finishes Errata 2.2.

Comment [m444]: Done

	integer that is the number of milliseconds from the epoch (positive or negative). It is a schema definition error if there is a specified length not equivalent to 8 bytes.
	binarySeconds and binaryMilliseconds may only be used when the type is xs:dateTime. (It is a schema definition error otherwise.)
	Annotation: dfdl:element, dfdl:simpleType
binaryCalendarEpoch	DateTime
	Used when dfdl:binaryCalendarRep is 'binarySeconds' or 'binaryMilliseconds'
	The epoch from which to calculate dates and times.
	If the time zone is omitted from the value, then UTC is used.
	Annotation: dfdl:element, dfdl:simpleType

Comment [SMH445]: Errata 2.57 allows unsigned integer **DONE**

Comment [SMH446]: Errata 2.56: Only for xs:dateTime.

DONE

Comment [SMH447]: Errata 2.53 allows time zone to be omitted

DONE

Examples of packed decimal format calendars for December 14, 1923 and calendarPattern of 'MMddyy' would be:

packed: (hexadecimal) `01 21 42 3C`,

bcd: (hexadecimal) `12 14 23`

ibm4690Packed: (hexadecimal) `12 14 23`

The 'C' nibble at the end of the 'packed' representation is a sign nibble, and the leading 0 nibble is just to align to a byte boundary..

Comment [SMH448]: Need to supply pattern for example – Mmddyy
(DONE)

Comment [m449]: Errata 2.2 asked for examples.
DONE

13.14 Properties Specific to Opaque Types (hexBinary)

There are no properties specific to opaque types

13.15 Nil Value Processing

Sometimes it is desirable to represent an unused element, place-holder for unknown information, or inapplicable information *explicitly* with an element, rather than by the lack of an element.

For example, it may be desirable to represent a sparsely populated array of data using a distinguished nil element to fill the locations where data is absent, thereby preserving the position for the elements that are present.

As another example, it may be desirable to represent an unused simple element by a value which is not conformant to the logical type of the element.

Such cases can be represented using the DFDL nil mechanism which is based on the XML Schema nil mechanism. DFDL provides what are commonly called "in-band" nil values by way of dfdl:nilKind='logicalValue', and also provides for two kinds of literal indicators of nil through dfdl:nilKind='literalValue' and dfdl:nilKind='literalCharacter'. Nil processing is used when the XSDL 'nillable' property of an element is true.

DFDL allows elements of complex type to be nillable. However, to avoid the concept of a complex element having a value, which does not exist in DFDL, the only permissible nil value is the empty string, represented by the DFDL %ES; entity and using dfdl:nilKind='literalValue'.

On parsing, an element occurrence is nil if the element has XSDL nillable 'true' and the data is a nil representation as defined in section 9.2.1. Specifically:

1. When dfdl:nilKind is 'literalValue', the **NilLiteralValue** region of the data stream matches any of the dfdl:nilValue values.

Comment [SMH450]: Entire section needs revisiting once action 140 agreed (DONE)

Comment [m451]: (DONE)

Comment [SMH452]: Is this the correct term ? (DONE)

Comment [m453]:

I think so: see http://publib.boulder.ibm.com/infocenter/iisinfsv/v8r7/index.jsp?topic=%2Fcom.ibm.swg.im.iis.ds.parjob.adref.doc%2Ftopics%2Fr_deeadvrf_The_modify_Operator_and_Nulls.html

Wording changed

DONE

Comment [m454]: Text Deleted

Comment [SMH455]: TODO: Add cross-ref

Comment [SMH456]: This will change to match new grammar.
DONE
SMH: Also added reference section 9.2.1.

2. When dfdl:nilKind is 'literalCharacter', all characters in the **NilLiteralCharacters** region of the data stream match the dfdl:nilValue character.

3. When dfdl:nilKind is 'logicalValue', the data contains a normal representation, and the **NilLogicalValue** region of the data stream, converted to the element's logical type, matches any of the dfdl:nilValue values.

For dfdl:nilKind='literalValue' or dfdl:nilKind='literalCharacter':

- Determination of whether the data is a nil representation for a literal nil happens first before any consideration of whether the representation is the empty, normal, or absent representations.

- Property dfdl:nilValueDelimiterPolicy controls whether matching one of the nil values also involves matching the initiator or terminator specified by the element. This gives control over whether a nil indicator may or may not also require the delimiters that a normal data element requires.

On unparsing, an element is nil if XSDL nillable is 'true' AND the element value is the distinguished value nil in the augmented infoset, in which case what is output to the data stream is one of the following:

1. When dfdl:nilKind is 'logicalValue' then the first value of dfdl:nilValue converted to the physical representation is output as the **NilLogicalValue** region.

2. When dfdl:nilKind is 'literalValue' then the first value of dfdl:nilValue is output as the **NilLiteralValue** region.

3. When dfdl:nilKind is 'literalCharacter' then the character from dfdl:nilValue, repeated to the needed length, is output as the **NilLiteralCharacters** region.

For dfdl:nilKind='literalValue' or dfdl:nilKind='literalCharacter' then dfdl:nilValueDelimiterPolicy determines whether any initiator or terminator also appear surrounding the literal nil in the output data.

13.16 Properties for Nillable Elements

These properties are used when the XSDL 'nillable' property of an element is 'true', and they control when and how the representation data are interpreted as having the logical meaning 'nil'.

Property Name	Description
nilKind	Enum
	Valid values 'literalValue', 'logicalValue', 'literalCharacter',
	Used when XSDL nillable is 'true' Specifies how dfdl:nilValue is interpreted to represent the nil value in the data stream..
	If 'literalCharacter' then dfdl:nilValue specifies a single character or a single byte that, when repeated to the length of the element, is the nil value. 'literalCharacter' may only be specified for fixed length elements, that is dfdl:lengthKind 'implicit' and 'explicit' when dfdl:length is not a DFDL expression, otherwise it is a schema definition error.
	If 'literalValue' then dfdl:nilValue specifies a list of DFDL literal strings that are the possible representations for nil.
	If 'logicalValue' then dfdl:nilValue specifies a list of logical values

Comment [SMH457]: Re-phrased in terms of the grammar.

Comment [SMH458]: Text Deleted

Comment [m459]: Text Deleted

Comment [m460]: Text Deleted

Comment [m461]: Text Deleted

Comment [SMH462]: Text Deleted

Comment [m463]: Text Deleted

	that are the possible logical values for nil.
	Complex elements can be nillable, but nilKind can only be 'literalValue' and nilValue must be "%ES;". It is a schema definition error otherwise.
	Annotation: dfdl:element(simpleType)
nilValue	List of DFDL String Literals, List of Logical Values, DFDL String Literal
	Specifies the text strings that are the possible literal or logical nil values of the element.
	If dfdl:nilKind is 'literalValue' then nilValue specifies a white space separated list of DFDL literal strings that are the possible representations for nil. On parsing the element value is nil if the trimmed data matches one of the literal strings in the list. On unparsing if the element value is nil the first nilValue from the list is output.
	If dfdl:nilKind is 'logicalValue' then nilValue specifies a white space separated list of logical values that are the possible logical values for nil. On parsing the element value is nil if the data, converted to its logical type, matches any of the logical values in the list. On unparsing if the element value is nil, the first nilValue from the list is converted to its physical representation and output.
	If dfdl:nilKind is 'literalCharacter' then nilValue specifies a single character or byte that, when repeated to the length of the element, is the nil representation. If a character, then it can be specified using a literal character or using DFDL entities. If a character is specified when dfdl:lengthUnits='bytes' then the nilValue must be a single-byte character. To specify a byte, it must be specified using a single "%#r;" entity. If a byte is specified when dfdl:lengthUnits='characters' then the encoding must be a fixed-width encoding.
	On parsing, the element value is nil if all characters in the untrimmed data content match the nilValue character . On unparsing, if the element value is nil the nilValue character is output to the needed length.
	There are restrictions on the string literal syntax of nilValue.
	When nilKind is literalValue and representation is text:
	o DFDL character entities are allowed
	o The raw byte entity (%#r) is allowed
	o DFDL Character classes (NL, WSP, WSP+, WSP*, ES) are allowed.
	When nilKind is literal value and representation is binary:
	o DFDL character entities are allowed
	o The raw byte entity (%#r) is allowed
	o DFDL Character class ES is allowed.
	o Other DFDL Character classes (NL, WSP, WSP+, WSP*) are not allowed.
	When nilKind is literalCharacter and representation is text:

Comment [SMH464]: Errata 2.20 clarifies allowable string literal content for literalValue

DONE

Comment [SMH465]: Errata 2.20 clarifies allowable string literal content for literalCharacter

DONE

Comment [SMH466]: Errata 2.8. **DONE**

Comment [SMH467]: Errata 2.20: Remove restriction that property only applies to text rep. **DONE**

Comment [SMH468]: Errata 2.8. **DONE**

	o DFDL character entities are allowed
	o The raw byte entity (%#r) is allowed.
	o DFDL Character classes (NL, WSP, WSP+, WSP*, ES) are not allowed.
	When nilKind is literalCharacter and representation is binary:
	o DFDL character entities are allowed
	o The raw byte entity (%#r) is allowed
	o DFDL Character classes (NL, WSP, WSP+, WSP*, ES) are not allowed.
	nilValue is sensitive to ignoreCase when nilKind is 'literalValue' or 'logicalValue', but not when nilKind is 'literalCharacter'
	Complex elements can be nillable, but nilKind can only be 'literalValue' and nilValue must be "%ES;". It is a schema definition error otherwise.
	Annotation: dfdl:element(simpleType)
nilValueDelimiterPolicy	Enum
	Valid values are 'none', 'initiator', 'terminator' or 'both'.
	Indicates that when the value nil is represented, an initiator (if one is defined), a terminator (if one is defined), both an initiator and a terminator (if defined) or neither must be present.
	Ignored if both dfdl:initiator and dfdl:terminator are "" (empty string).
	Ignored if nilKind is set to 'logicalValue' In this case the DFDL processor treats a nil representation like any other representation of the element in that it expects delimiters when parsing, outputs them when unparsing.
	'initiator' indicates that, on parsing, the dfdl:initiator followed by one of the dfdl:nilValue is the necessary syntax to indicate that a nil representation is present. It also indicates that on unparsing when the logical value is nil that the dfdl:initiator will be output followed by the first of the dfdl:nilValue.
	'terminator' indicates that, on parsing, one of the dfdl:nilValue followed by the dfdl:terminator is the necessary syntax to indicate that a nil representation is present. It also indicates that on unparsing when the logical value is nil the first of the dfdl:nilValue followed by the dfdl:terminator will be output.
	'both' indicates that, on parsing, both the dfdl:initiator and dfdl:terminator must be present with one of the dfdl:nilValue to indicate that a nil representation is present. On unparsing the dfdl:initiator followed by the first dfdl:nilValue, followed by the dfdl:terminator will be output.
	'none' indicates that one of the dfdl:nilValue without any dfdl:initiator or dfdl:terminator triggers creation of a nil value. On unparsing the first of the dfdl:nilValue is output without the dfdl:initiator or dfdl:terminator.
	It is a schema definition error if dfdl:nilValueDelimiterPolicy is set to 'none' or 'terminator' when the parent xs:sequence has

Comment [m469]: Errata 2.20 (the rest of it) Done

Comment [SMH470]: Errata 2.71. Clarifies ignoreCase sensitivity for each enum

DONE

Comment [SMH471]: Errata 3.10 adds nillable complex elements

DONE

Comment [SMH472]: Errata 2.58: Not applicable when logicalValue

DONE

	dfdl:initiatedContent 'yes'.
	Annotation: dfdl:element(simpleType)

13.17 Properties for Element Defaults Control

The DFDL element defaults processing uses XSDL default, XSDL fixed or dfdl:useNilForDefault to provide a default value. See section 13.15 Nil Value Processing for a full description.

Comment [SMH473]: OPEN: Update to 9.4

Property Name	Description
useNilForDefault	Enum
	Valid values are 'yes', 'no'
	When the conditions for applying a simple element default are satisfied, this property controls whether to use the special value nil or to use the XSDL default or fixed properties.
	This property has precedence over the XSDL default and XSDL fixed properties. It is only used, and must be defined, if the XSDL:nillable property is 'true'.
	Defaulting occurs as described in section 9.4 with nil as the default value. The dfdl:nilValue property must specify at least one nil value otherwise it is a schema definition error. The dfdl:nilKind property may be any of its values.
	Annotation: dfdl:element (simpleType)

Comment [SMH474]: Errata 2.45: Only applicable if nillable is true, and must be set if so

DONE

Comment [SMH475]: "It is only used, and must be defined, if …"

DONE

Comment [SMH476]: OPEN – make into cross-reference

14. Sequence Groups

The following properties are specific to sequences.

Property Name	Description
sequenceKind	Enum Valid values are 'ordered', 'unordered' When 'ordered', this property means that the contained items of the sequence will be encountered in the same order that they appear in the schema, which is called schema-definition-order. When 'unordered', this property means that the items of the sequence will be encountered in any order. Repeating occurences of the same element do not need to be contiguous. The children of an unordered sequence MUST be xs:element otherwise it is a schema definition error. Annotation: dfdl:sequence, dfdl:group (sequence)
initiatedContent	Enum Valid values are 'yes', 'no' When 'yes' indicates that all the children of the sequence are initiated. It is a schema definition error if any children have their dfdl:initiator property set to the empty string. If the child is optional then it is deemed to have been found when its initiator has been found. Any subsequent error parsing the child will not cause the parser to backtrack to try other alternatives. When 'no', the children of the sequence may have their dfdl:initiator property set to the empty string. Annotation: dfdl:sequence, dfdl:choice, dfdl:group

A sequence can have an initiator and/or a terminator as described earlier.

14.1 Empty Sequences

A sequence having no children is syntactically legal in DFDL. In the data stream, such a sequence can have non-zero length **LeftFraming** and **RightFraming** regions, but the SequenceContent region in between must be empty. It is a processing error if the SequenceContent region of an empty sequence has non-zero length when parsing.

XML schema does not define an empty sequence that is the content model of a complex type definition as effective content so any DFDL annotations on such a construct would be ignored. It is a schema definition error if the empty sequence is the content model of a complex type, or if a complex type has nothing in its content model at all.

> **Comment [SMH477]:** Errata 2.48: **DONE**

14.2 Sequence Groups with Separators

Additional properties apply to sequence groups that use text delimiters to separate one occurrence of a member of the group from the next. Such a delimiter is called a separator. DFDL provides several properties that control the parsing and writing of separators, and satisfy the requirement to model sequences where:

1. A separator has alternative potential representations in the data.
2. A separator is placed before, after, or between occurrences in the data.
3. Separators are used to indicate the position of occurrences in the data

These requirements are addressed by the properties dfdl:separator, dfdl:separatorPosition and dfdl:separatorSuppressionPolicy, as described below.

These properties combine to define the syntax for a sequence group with dfdl:sequenceKind 'ordered'. Not all combinations of the properties will give rise to consistent syntax, so some combinations are disallowed and will give rise to a schema definition error.

In some sequences, the presence of separators alone is enough to establish the identification of occurrences within the sequence. Such a sequence is called a *positional* sequence.

Positional sequence - Each occurrence in the sequence can be identified by its position in the data. Typically the components of such a sequence do not have an initiator. In some such sequences, the separators for optional zero-length occurrences may or must be omitted when at the end of the group. A positional sequence can be modelled by setting dfdl:separatorSuppressionPolicy to 'required', 'trailingEmptyStrict' or 'trailingEmpty'.

Non-positional sequence - Occurrences in the sequence cannot be identified by their position in the data alone. Typically the components of such a sequence have an initiator. Such sequences allow the separator to be omitted for optional zero-length occurrences anywhere in the sequence. Speculative parsing and backtracking must be used to identify each occurrence. A non-positional sequence can be modelled by setting dfdl:separatorSuppressionPolicy to 'anyEmpty'.

> **Comment [SMH478]:** Added as per errata addendum. (DONE)

Property Name	Description
separator	List of DFDL String Literals or DFDL Expression
	Specifies a whitespace separated list of alternative literal strings that are the possible separators between a sequence of elements or multiple occurrences of an element.
	This property can be computed by way of an expression which returns a string of whitespace separated values. The expression must not contain forward references to elements which have not yet been processed.
	The **Separator, PrefixSeparator** and **PostfixSeparator** regions contain one of the strings specified by the dfdl:separator property. When this property has "" (empty string) as its value then the separator region is of length zero. It is not permitted for an expression to return an empty string. That is a schema definition error.
	When parsing, the list of values is processed in a greedy manner, meaning it takes all the separators, 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 separator with the longest match is the one that is selected as having been 'found', with length-ties being resolved so that the matching

> **Comment [SMH479]:** Errata 2.63. **DONE**

	separator is selected that is first in the order written in the schema. Once a matching separator is found, no other matches will be subsequently attempted (ie, there is no backtracking).
	On unparsing the first separator in the list is used as the separator.
	If a child element uses an escape scheme, then the escape scheme also applies to any separator.
	If dfdl:ignoreCase is 'yes' then the case of the string is ignored by the parser.
	Annotation: dfdl:sequence, dfdl:group (sequence)
separatorPosition	Enum
	Valid values 'infix', 'prefix', 'postfix'
	'infix' means the separator occurs between the elements in the **Separator** grammar region.
	'prefix' means the separator occurs before each element in the **Separator** grammar region and the **PrefixSeparator** grammar region.
	'postfix' means the separator occurs after each element in the **Separator** grammar region and the **PostfixSeparator** grammar region.
	Annotation: dfdl:sequence, dfdl:group (sequence).
separatorSuppressionPolicy	Enum
	Valid values 'never', 'anyEmpty', 'trailingEmpty', 'trailingEmptyStrict'
	Only applicable if separator is not "" (empty string) and dfdl:sequenceKind is 'ordered'.
	Controls the circumstances when separators are expected in the data when parsing, or generated when unparsing, if an optional element occurrence or a group has a zero-length representation.
	See section 0
	Separators and Suppression.
	When dfdl:sequenceKind is 'unordered' then 'anyEmpty' is implied.
	Annotation: dfdl:sequence, dfdl:group (sequence)

> **Comment [m480]:** I revised the wording of this somewhat to reflect that the longest-match behavior is not a static sort of the string literal tokens, but it is the longest match against the actual data.
> (DONE)

> **Comment [SMH481]:** Errata 2.70 specifies the parsing matching algorithm
>
> DONE

> **Comment [SMH482]:** Errata 3.14 changes the name of the property and its enums
>
> DONE

> **Comment [SMH483]:** Text Deleted

> **Comment [m484]:** (DONE) Errata 3.14.

> **Comment [m485]:** Errata 3.16 Done

14.2.1 Separators and Suppression

When parsing a sequence group that specifies a separator, the number of occurrences and separators that are expected in the data stream for a child element depends on several factors:

- Whether occurrences are optional or required
- Whether the occurrences have a zero-length representation
- Whether occurrences are trailing
- Whether the sequence is positional

- The dfdl:occursCountKind of the element

Where to expect a separator for optional content of zero-length is controlled by property dfdl:separatorSuppressionPolicy.

separatorSuppressionPolicy	Implications
never	Positional sequence where all occurrences MUST be found in the data, along with their associated separator.
trailingEmptyStrict	Positional sequence where *trailing occurrences* that have zero length representation MUST be omitted from the data, along with their associated separator.
trailingEmpty	Positional sequence where *trailing occurrences* that have zero length representation MAY be omitted from the data, along with their associated separator.
anyEmpty	Non-positional sequence where any occurrences that have zero length representation MAY be omitted from the data, along with their associated separator. It must be possible for speculative parsing to identify which elements are present.

Table 20 Sequence groups and separator suppression

The following are definitions for terminology used in this section:

Potentially trailing element – An array or optional element describes an occurrence that is said to be *potentially trailing* if the element is capable of having a zero length representation and is followed in its enclosing group definition by only additional potentially trailing elements or potentially trailing groups. Intuitively, the array or optional element *could* be last.

Potentially trailing group – A group is said to be *potentially trailing* if the group has no framing and contains only potentially trailing element declarations/references, or recursively similar sequence or choice groups, and is followed in its enclosing group definition by only additional potentially trailing elements or potentially trailing groups.

Trailing or Actually Trailing – An element occurrence or group occurrence in the data is said to be actually *trailing* if it is potentially trailing and has zero-length representation and is not followed in the data by any other non-zero length element occurrence or group occurrence.

In the sections that follow, it is important to remember that the dfdl:separatorSuppressionPolicy property is carried on the sequence, while the XSDL minOccurs, XSDL maxOccurs and dfdl:occursCountKind properties are is carried on an *element* in that sequence.

14.2.2 Parsing

When an element is required and is not an array then one occurrence is always expected along with its separator. The dfdl:separatorSuppressionPolicy is not applicable and the implied behavior is 'never'.

Otherwise the behaviour is dependent on dfdl:occursCountKind.

When dfdl:occursCountKind is 'fixed' then XSDL minOccurs must equal maxOccurs and that many occurrences are always expected along with their separators. The dfdl:separatorSuppressionPolicy is not applicable and the implied behaviour is 'never'.

When dfdl:occursCountKind is 'expression' the number of occurrences is given by dfdl:occursCount and exactly that many occurrences are always expected along with their separators. The dfdl:separatorSuppressionPolicy is not applicable and the implied behaviour is 'never'.

When dfdl:occursCountKind is 'parsed' any number of occurrences and their separators are expected. The dfdl:separatorSuppressionPolicy is not applicable and the implied behaviour is 'anyEmpty'.

When dfdl:occursCountKind is 'stopValue', any number of occurrences and their separators are expected followed by the stop value and its separator. The dfdl:separatorSuppressionPolicy is not applicable and the implied behaviour is 'anyEmpty'.

When dfdl:occursCountKind is 'implicit', between XSDL minOccurs and maxOccurs (inclusive) occurrences and their separators are expected. The dfdl:separatorSuppressionPolicy is applicable and determines when separators are expected for optional zero-length occurrences.

The behaviour for 'implicit' is more fully expressed in matrix form. The cells in the matrix give the number of occurrences of element values that are expected in the data stream when parsing, for the different values of dfdl:separatorSuppressionPolicy. The number of occurrences also depends whether XSDL maxOccurs is unbounded or not, and the position of the element in the sequence. The number of separators can be inferred from this, taking into account dfdl:separatorPosition.

separatorSuppressionPolicy	occursCountKind implicit					
	Potentially Trailing				Not Potentially Trailing	
	maxOccurs unbounded		maxOccurs bounded			
	element not declared last	element declared last	element declared last, or occurrence followed by end-of-group	element not declared last, and occurrence not followed by end-of-group	maxOccurs unbounded	maxOccurs bounded
never	schema definition error	RepDef(min) ~ Rep(M < INF) ~ RepNonZero(1)]	RepDef(min) ~ Rep(max - min)		schema definition error	RepDef(min) ~ Rep(max - min)
trailingEmptyStrict	schema definition error	RepDef(min) [~ Rep(M < INF) ~ RepNonZero(1)]	RepDef(min) [~ Rep(M < max - min) ~ RepNonZero(1)]	RepDef(min) ~ Rep(M <= max - min)	schema definition error	RepDef(min) ~ Rep(max - min)
trailingEmpty		RepDef(min) ~ Rep(M < INF)				
anyEmpty		RepDef(min) ~ Rep(M < INF)	RepDef(min) ~ Rep(M <= max - min)		RepDef(min) ~ Rep(M < INF)	RepDef(min) ~ Rep(M <= max - min)

<Add table number>

Cells that share a common behavior that are adjacent have been merged.

The notation in each cell uses the "~" symbol to mean "followed by" in the data stream. Square brackets surround things that are optional, as in they may or may not appear in the data stream.

The descriptions found in the cells of the matrix do not provide a parsing algorithm, but rather state declaratively a pattern that the data must match in order to be correctly parsed.

RepDef(min) is short for "representation" and "defaultable". It means XSDL minOccurs occurrences of nil, empty or normal representation[28]. These are required occurrences so default rules apply for empty representations. XSDL minOccurs may be 0, in which case there are no required occurrences.

Rep(M) means M occurrences of nil, empty, normal or absent representation. These are optional occurrences so default rules do not apply for empty representations.

RepNonZero(1) means an occurrence of a nil, empty or normal representation where such a representation does not have zero-length[29]. This is an optional occurrence so default rules do not apply.

A notation like Rep(M <= max – min) means that there are M occurrences, where M is some value between the values of the XSDL minOccurs and maxOccurs properties. When an unbounded number of occurrences is possible this is shown explicitly by Rep(M < INF) , INF meaning infinity or unbounded.

14.2.2.1   Errors when maxOccurs='unbounded'

In the matrix above we see that there are some situations where XSDL maxOccurs 'unbounded' simply doesn't make sense when combined with dfdl:separatorSuppressionPolicy.

It is a schema definition error if a sequence has dfdl:separatorSuppressionPolicy 'never' and a child element has dfdl:occursCountKind 'implicit' and XSDL maxOccurs 'unbounded'.

It is a schema definition error if a sequence has dfdl:separatorSuppressionPolicy 'trailingEmptyStrict' or 'trailingEmpty', and a child element has dfdl:occursCountKind 'implicit' and XSDL maxOccurs 'unbounded' and either the child element cannot have potentially trailing occurrences or the child element can have potentially trailing occurrences but the element is not declared last in the sequence.

14.2.2.2   Example Scenarios for Separator Suppression

Consider the cell of the matrix above for an element declaration of simple type xs:int, with maxOccurs='5', minOccurs='0', which is potentially trailing, element declared last or occurrence followed by end-of-group, and dfdl:separatorSuppressionPolicy='trailingEmptyStrict'. The cell contains this description:

RepDef(min) [~ Rep(M < max – min) ~ RepNonZero(1)]

Also assume dfdl:emptyValueDelimiterPolicy='both', dfdl:initiator='[', dfdl:terminator=']' and the enclosing sequence has dfdl:separator='|' (pipe) with dfdl:separatorPosition='infix'. Finally, there is a default value 0 defined, and the element is not nillable.

Since minOccurs='0', the first term, RepDef(min) vanishes. So we have left

Rep(M < max - min) ~ RepNonZero(1).

Rep(M) means we can have absent representations, and they will simply be omitted from the infoset.

So this data

---

[28] Absent representation implies processing error for 'implicit' when less than or equal to minOccurs.

[29] Absent representation always implies zero-length. Nil, empty, and normal representations can also be zero-length with the right combinations of properties. See Section (TBD: Cross Reference).

Comment [m491]: What does 'if permitted' mean here? When could minOccurs="0" not be permitted? Shouldn't this just say "when minOccurs="0"?

DONE

SMH: Yes – I think the 'permitted' is a hang over for when the table also applied to 'fixed'.

Comment [m492]: (OPEN) Provide some motivating example or discussion. Otherwise this just seems too arbitrary.

Comment [m493]: (OPEN) Does this rule make it impossible to just put dfdl:separatorSuppressionPolicy='trailingEmpty' into the scope, and have that apply everywhere that there are trailing empty situations? I think that is very desirable.

I think this rule will cause SDEs on some elements if you just put 'trailingEmpty' or 'trailingEmptyStrict' into the default scope.

Perhaps we have an option to simply say in this case the separatorSuppressionPolicy is ignored and nothing is suppressed for that element?

SMH: I think it is ok to say that separatorSuppressionPolicy is ignored if a sequence is unordered or if OCK is not 'implicit' but beyond that it's getting too subtle. If you are modeling separators and using 'implicit' you need to be absolutely clear what you are saying for separator suppression on the sequence, and this rule forces you to think about the implications of making an element unbounded.

Comment [m494]: (OPEN) see comment above about RepNonZero(1) – 1 is too restrictive

```
[1]|[2]|[3]|[4]|[5]
```

parses and 5 items appear in the infoset.

This data

```
|||[4]
```

also parses because absent representations are accepted, but only one item appears in the infoset.(The fact that the occurrence was fourth in the array is not preserved into the Infoset). However this data

```
|||[4]|
```

causes a processing error because there is a final trailing separator.

Now consider the same scenario but minOccurs='2'. The first term reappears as RepDef(2). The data

```
|||[4]
```

which previously parsed successfully would now cause a processing error because the first two occurrences are required, so they have to be either a normal representation, that is, matching xs:int syntax with surrounding initiator and terminator, or the empty representation which is `[]`. An example which will parse correctly with minOccurs='2' is:

```
[1]|[]||[4]
```

In this case the Infoset will contain 3 items with values 1, 0, 4. The 0 value arises because the occurrence has the empty representation, the occurs index is 2 so it is required, and there is a default value 0.

If the scenario is changed so that dfdl:separatorSuppressionPolicy= 'trailingEmpty' then a different cell applies.

RepDef(min) [~ Rep(M < max – min)]

This has a more lax behavior so that this data is also acceptable:

```
[1]|[]||[4]|
```

In this case the final trailing separator is tolerated, though when unparsing this final trailing separator would not be created. This is a case where what is parsed will not be exactly recreated on unparsing from the resulting infoset, but all the information content is preserved.

Now consider the same scenario but maxOccurs='unbounded'. In that case this data is acceptable:

```
[1]|[]||||||||||||||||||||||||[4]|||||||||||||||||||||
```

The infoset values are again 1, 0, 4. But all these excess separators are tolerated.

> **Comment [m495]:** (OPEN) Extend scenario to show using nillable to preserve index positions

> **Comment [m496]:** (OPEN) Extend scenario to show how much more rigid anyEmpty and never suppression are.

14.2.3 Unparsing

When an element is required and is not an array then one occurrence is always output along with its separator. The dfdl:separatorSuppressionPolicy is not applicable and the implied behaviour is 'never'.

Otherwise the behaviour is dependent on dfdl:occursCountKind.

When dfdl:occursCountKind is 'fixed' or 'expression' the occurrences in the augmented Infoset are always output along with their separators. The dfdl:separatorSuppressionPolicy is not applicable and the implied behaviour is 'never'.

When dfdl:occursCountKind is 'parsed' non zero-length occurrences in the augmented Infoset are output along with their separators. The dfdl:separatorSuppressionPolicy is not applicable and the implied behaviour is 'anyEmpty'.

When dfdl:occursCountKind is 'stopValue' non zero-length occurrences in the augmented Infoset are output along with their separators followed by the stop value and its separator. The dfdl:separatorSuppressionPolicy is not applicable and the implied behaviour is 'anyEmpty'.

When dfdl:occursCountKind is 'implicit' the occurrences in the augmented Infoset are output along with their separators. The dfdl:separatorSuppressionPolicy is applicable and helps determine whether optional zero length occurrences and their separators are output.

The behaviour for 'implicit' is more fully expressed in matrix form. The cells in the matrix give the number of occurrences of element values that are output to the data stream when unparsing, for the different values of dfdl:separatorSuppressionPolicy. The number of occurrences also depends whether XSDL maxOccurs is unbounded or not, and the position of the element in the sequence. The number of separators output can be inferred from this, taking into account dfdl:separatorPosition.

separatorSuppression Policy	occursCountKind implicit					
	Potentially Trailing				Not Potentially Trailing	
	maxOccurs unbounded		maxOccurs bounded		maxOccurs unbounded	maxOccurs bounded
	element not declared last	element declared last	element declared last, or occurrence followed by end-of-group	element not declared last, and occurrence not followed by end-of-group	maxOccurs unbounded	maxOccurs bounded
never			unparse N occurrences ~ unparse (maxOccurs − N) trailing zero-length occurrences	unparse N occurrences ~ unparse (maxOccurs − N) trailing zero-length occurrences		unparse N occurrences ~ unparse (maxOccurs − N) trailing zero-length occurrences
trailingEmptyStrict	schema definition error	unparse N occurrences (suppressing trailing zero-length reps)			schema definition error	
trailingEmpty						
anyEmpty	unparse N occurrences (suppressing any optional zero-length occurrences)					

<Add table number>

Cells that share a common behavior that are adjacent have been merged.

The notation in each cell uses the "~" symbol to mean "followed by" in the output data stream.

N stands for the number of elements in the *augmented* Infoset, which includes any defaults.

unparse N occurrences means output N unparsed infoset items and associated separators.

unparse(M) trailing zero length occurrences means output M adjacent separators (according to dfdl:separatorPosition) as if separating M element occurrences.

(suppressing trailing zero-length reps) implies the unparser must look ahead into the infoset and determine when the representations will be zero-length, and then identify those in trailing position. No separators are output corresponding to the trailing zero-length representations.

14.2.4   Round Trip Ambiguities

The overlapping nature of the possible representations: normal, empty, nil, and absent as well as the behavior of separator suppression create a number of ambiguities where taking an Infoset, unparsing it, and reparsing it will result in a second Infoset that is not the same as the original. However taking the second Infoset, unparsing it, and reparsing it, will result in a third Infoset which is the same as the second.

When unparsing, if a string Infoset item happens to contain a string that matches either one of the nilValues or the default value, it is not given any special treatment. The string's characters are output, or if the value is the empty string, zero length content is output. (In both cases along with an initiator or terminator if defined.) This creates an ambiguity where one can unparse an Infoset item which is not the special value *nil*, but when reparsed will produce *nil* in the Infoset.

These ambiguities are natural and unavoidable. If the nilValue is the 3-character string "nil", then encountering the characters "nil" in the data stream will parse to produce the special value *nil* in the Infoset. If you unparsed a string infoset item with contents of the 3 characters "nil", this will be output as the letters "nil", which on parse will not produce a string with the characters "nil", but rather the special value *nil* in the Infoset.

To avoid this issue, one can use validation, along with a pattern that prevents the string from matching any of the nil values.

Similarly, for some formats that use separators, when unparsing and there is no Infoset item, the unparser may still output a zero-length representation meaning the element is optional and not present. For example, dfdl:separatorSuppressionPolicy='never' will cause this behavior. In this situation, one can unparse an Infoset where there is no Infoset item, but reparsing that data will create an Infoset item with special value *nil* or an empty string.

Example: A nillable optional array element with occursCountKind 'implicit' and %ES; is the first nilValue, within a separated sequence with dfdl:separatorSuppressionPolicy "never", but not potentially trailing. If there are less than maxOccurs items in the Infoset, separators will be output up to maxOccurs with zero length between the separators. On parsing, those zero lengths will be interpreted as *nil*, so the array element will always have maxOccurs Infoset items, some of which will be *nil*.

> **Comment [SMH500]:** TODO
> Move section elsewhere (OPEN)

14.3 Unordered Sequence Groups

In DFDL, ordered and unordered are characteristics of the representation only. Logically, sequence groups are always in schema order.

The semantics of unordered groups (sequence with dfdl:sequenceKind='unordered' property) are expressed by way of a source-to-source transformation of the declaration, and by a data transformation on the resulting infoset. An implementation may use any technique consistent with this semantic.

The source to source transformation turns the declaration of an unordered group into an ordered sequence that contains an array element that contains a choice. Each element declaration of the unordered group becomes an alternative element within the choice. The unordered group's separator and terminator become the Separator and Terminator of the surrounding sequence. The dfdl:sequenceKind property is dropped, but other DFDL annotation properties are preserved. The XSDL maxOccurs and minOccurs on any element of the unordered sequence are dropped when the element is placed into the choice.

For example:

```
<xs:sequence dfdl:sequenceKind="unordered">
  <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 is conceptually rewritten into an element declaration and reference like so:

```
<xs:element name="dummy">
  <xs:complexType>
    <xs:choice>
      <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 dfdl:sequenceKind="ordered">
  <xs:element ref="dummy" minOccurs="1" maxOccurs="12"/>
</xs:sequence>
```

Schema definition errors are then detected as for choice group types. Notice how the XSDL minOccurs and maxOccurs for the rewritten element reference are computed based on the possible occurrences from the original source.

Processing then constructs this array element by parsing the data.

The post processing then transforms this array back into the original sequence of non-choice elements. That is, the array is then used to populate 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>
```

This is a logical-value to logical-value transformation. Ordered and unordered are characteristics of the representation only. The transformation here is the obvious one where all array elements having the first choice alternative as their value are accumulated into the first child element of the logical sequence. If there is either no such value or more than one such value, then the first child element must be an array or optional declaration (appropriate XSDL minOccurs and maxOccurs) so that it can accommodate the number of values found. The dimensionality of the first element must accommodate the number of values actually found. It is a processing error if it cannot. This algorithm repeats for the array elements having the 2nd choice alternative as their value, and so on until all the choice alternative values have been moved into their corresponding elements/arrays in the logical sequence group, and all logical sequence elements have been populated in a manner conforming to their XSDL minOccurs constraints.

An unordered sequence is of fixed length if the same sequence is fixed length when the unordered property is removed.

On Unparsing, the behavior is exactly as if dfdl:sequenceKind='ordered'. That is, the elements are output in schema declaration order.

14.4   Floating Elements

Elements within an ordered sequence can be designated as floating which means that they can appear in any position within the sequence.[30]

Property Name	Description
floating	Enum
	Valid values are 'yes', 'no'
	Whether the occurrences of an element in an ordered sequence can appear out-of-order in the representation.
	When parsing, and dfdl:floating is 'yes', occurences of the element may be encountered in the representation in any position within its containing sequence. If present they are placed into the infoset in schema declaration order. If the element repeats, occurences do not need to be contiguous in the representation.
	When parsing, and dfdl:floating is 'no', occurences of the element must be in schema declaration order, and, if present, they are placed into the infoset in schema declaration order. It is a processing error if instances of the element are not encountered in schema declaration order.
	When unparsing, occurrences of the element are expected in the infoset in schema declaration order, and are output in the representation in schema declaration order. It is a processing error if occurrences of the element are not encountered in schema declaration order,
	It is a schema definition error if an unordered sequence or a choice contains any element with dfdl:floating='yes'.
	It is a schema definition error if an ordered sequence contains any element with dfdl:floating='yes' and also contains non-element component (such as a choice or sequence model group).
	Annotation: dfdl:element

An ordered sequence with floating components is similar to an unordered sequence except only the floating elements may be out of order.

An ordered sequence of n element children with either n or n-1 of those children with dfdl:floating='yes' is equivalent to an unordered sequence with the same n element children with dfdl:floating='no'.

A complex element with dfdl:floating='yes' can have as its content model a sequence with elements that also have dfdl:floating='yes'.

This makes every element in a sequence containing one or more floating elements a point of uncertainty, similar to the way every element in an unordered sequence is a point of uncertainty.

---

[30] . The NTE segment in the X12 EDI standard is an example of a floating element.

A parser MUST look for the element defined at that position in the schema first and then look for the floating elements in the order they are defined in the schema.

14.5 Hidden Groups

Some fields in the physical stream provide information about other fields in the stream and are not really part of the data. For example, a field could give the number of repeats in a following array. These fields may not be of interest to an application so may be removed from the Infoset on parsing by marking them as hidden. A hidden sequence group allows fields to be defined that will not be added to the Infoset on parsing and will not be expected in the Infoset on unparsing.

```
<xs:element name="root">
  <xs:complexType>
    <xs:sequence>

      <xs:sequence>
        <dfdl:sequence hiddenGroupRef="tns:hiddenRepeatCount">
      </xs:sequence>

      <xs:element name="arrayElement" type="xs:int"
                  minOccurs="0" maxOccurs="unbounded"
                  dfdl:occursCountKind="expression"
                  dfdl:occurCount= "{../repeatCount}"
                  dfdl:representation="binary" dfdl:lengthKind="implicit" />
    </xs:sequence>
  </xs:complexType>
</xs:element>

<xs:group name="hiddenRepeatCount" >
  <xs:sequence>
    <xs:element name="repeatCount" type="xs:int"
                dfdl:outputValueCalc="{count(../arrayElement)}"
                dfdl:representation="binary" dfdl:lengthKind="implicit" />
  </xs:sequence>
</xs:group>
```

Hidden elements within a hidden group can be referenced via path expressions using the same DFDL expression that would be used if it were not hidden. To clarify, their names are hidden, but they occupy numeric locations normally as children of their parent element.

Hidden elements can (typically will) contain the regular DFDL annotations to define their physical properties and on unparsing to set their value. They are processed using the same behavior as non-hidden elements.

When the dfdl:hiddenGroupRef property is specified, all other DFDL properties are ignored. It is a schema definition error if the sequence is not empty. It is a schema definition error if the sequence is the only thing in the content model of a complex type definition. It is a schema definition error if dfdl:hiddenGroupRef appears on a xs:group reference, that is,unlike most format properties that apply to sequences, dfdl:hiddenGroupRef cannot be combined from a xs:group reference. .

A hidden sequence may appear within another hidden sequence.

Property Name	Description
hiddenGroupRef	QName
	Reference to a global model group definition that defines the hidden element or elements.
	The model group within the model group definition may be a xs:sequence or xs:choice
	It is a schema definition error if the value is the empty string.

Comment [m501]:
DONE

Clarification: do elements within hidden groups take up numeric spots in the infoset. I.e., if a hidden group has one element in it, and appears before any other non-hidden child elements, does the hidden element take up the first child position, or does it magically disappear from the infoset and not have a numeric position?

This statement that says you can use the same path expressions implies that hidden elements take up numeric positions because numeric child indexing can be used (within the filter of an expression, i.e. foo/bar[2].)

SMH: They take up numeric spots, as the statement implies

Comment [SMH502]: By implication from section 14.1.
DONE – clarified wording

Comment [m503]: (OPEN)
So, should this be removed then? It's not incorrect.
DONE – clarified wording

Comment [SMH504]: xs:group or xs:group reference, (for clarity).
DONE

Comment [SMH505]: Errata 3.1 clarification
DONE

Comment [SMH506]: Errata 3.1 changes this to disallow empty string and behave like dfdl:ref

DONE

Comment [SMH507]: No, empty string is a schema definition error

DONE

| | It is not possible to place this property in scope on a dfdl:format annotation. Annotation: dfdl:sequence |

Table 21 Hidden properties

15. Choice Groups

The following properties are specific to xs:choice.

Property Name	Description
choiceLengthKind	Enum
	Valid values are 'implicit', 'explicit'
	'implicit' means the branches of the choice are not filled, so the ChoiceContent region is variable length depending on which branch appears.
	'explicit' means that the branches of the choice are always filled to the fixed length specified by dfdl:choiceLength, so the ChoiceContent region is fixed length regardless of which branch appears.
	Annotation: dfdl:choice, dfdl:group (choice)
choiceLength	Integer
	Only used when dfdl:choiceLengthKind is 'explicit'.
	Specifies the length of the choice in bytes, so the ChoiceContent region is fixed length regardless of which branch appears. A *ChoiceUnused* region is therefore possible which when unparsing is filled with dfdl:fillByte.
	Annotation: dfdl:choice, dfdl:group (choice)
initiatedContent	Enum
	Valid values are 'yes', 'no'
	When 'yes' indicates that all the branches of the choice are initiated. It is a schema definition error if any children have their dfdl:initiator property set to the empty string. The branch is deemed to have been found when its initiator has been found. Any subsequent error parsing the branch will not cause the parser to backtrack.
	When 'no', the branches of the choice may have their dfdl:initiator property set to the empty string.
	Annotation: dfdl:sequence, dfdl:choice, dfdl:group
choiceBranchRef	DFDL Expression
	The expression must evaluate to an xs:string which must not be the empty string.
	This property is used only when parsing.
	The resultant string must match (case insensitive) the dfdl:elementID property value of one of the element branches of the choice. If so, it discriminates to that branch. The parser then goes straight to that branch, ignoring consideration of any other choice branches. No backtracking of this decision occurs if there is a subsequent processing error.
	It is a processing error if the value of the expression does not match one of the elementIDs for the branches.
	When choiceBranchRef is present, all choice branches must be local elements or element references. It is a schema definition error otherwise.

Comment [SMH508]: Errata 2.55. **DONE**

	It is not possible to place this property in scope on a dfdl:format annotation.
	Annotation: dfdl:choice
elementID	DFDL String Literal
	This literal provides an alternate identifier for an element. When the choiceBranchRef expression evaluates to a string matching this property's value, the choice is discriminated to this element.
	It is a schema definition error if individual elementID values are not unique across all elements that are branches of a choice that carries choiceBranchRef, or if the elementId values of global elements are not unique within a given namespace.
	Raw byte entities and character classes are not allowed.
	This property is only used when parsing.
	It is not possible to place this property in scope on a dfdl:format annotation.
	Annotation: dfdl:element

A choice can have an initiator and/or a terminator as described earlier.

We will use this terminology:

Branch	A *branch* is one of the available alternatives within a choice. A branch can be an element of simple type or complex type, or it can be an embedded sequence, choice or group reference.
Root of the Branch	Each *branch* conceptually has a single element, sequence, choice or group reference component at its root. This element is known as the *Root of the Branch*.

Table 22 Choice group terminology

The Root of the Branch MUST NOT be optional. That is XSDL minOccurs MUST BE greater than 0.

When processing a choice group the parser validates any contained path expressions. If a path expression contained inside a choice branch refers to any other branch of the choice, then it is a schema definition error. Note that this rule handles nested choices also. A path that navigates outward from an inner choice to another alternative of an outer choice is violating this rule with respect to the outer choice.

15.1   Resolving Choices

A choice corresponds to concepts called variant records, multi-format records, discriminated unions, or tagged unions in various programming languages. In some contexts choices are referred to generally as 'unions'. However, this should not be confused with XML schema unions.

When processing a choice, there are two ways to resolve the intended branch. In one, speculative parsing is used. In the other, a constant-time direct dispatch to a branch is performed.

15.1.1   Resolving Choices via Speculation

Speculative resolution works as follows:

1. Attempt to parse the first branch of the choice.
2. If this fails with a processing error
    a. If a dfdl:discriminator evaluated to true earlier on this branch
        then the parser is 'bound' to this choice and parsing of the entire choice construct
        fails with a processing error.
    b. If a dfdl:discriminator has not evaluated to true then we repeat from step 1 for the
        next branch of the choice.
3. It is a processing error if the branches of the choice are exhausted.
4. If a branch is successfully parsed without error, then that branch's infoset becomes the
    infoset for the parse of the choice construct.

It is not possible for variable settings to be communicated from the speculative attempt to parse a
branch to any other parsing situation. The speculative effort is completely isolated. Whether it
succeeds or fails, neither the parse position in the source data, nor anything in the variable
memory, nor the infoset is affected.

Nested choices can require unbounded look ahead into the data.

15.1.2 Resolving Choices via Direct Dispatch

Direct dispatch provides a constant-time dispatch to a choice branch independent of how many
choice branches there are.

Direct dispatch is indicated by the choiceBranchRef property. This expression is evaluated to
compute the element identifier matching (case insensitive) the dfdl:elementID property of one of
the choice branches, all of which must be local element declarations or element references.

When a match is found, it is as if a discriminator had evaluated to true on that branch. It is
selected as resolution of the choice, and there is no backtracking to try other alternative
selections.

The elementID property can be placed on global element declarations, element references, or
local element declarations. It must be unique within one choice when placed on element
references or local element declarations. It must be unique within the namespace if placed on
global element declarations.

Note that it is a schema definition error if both initiatedContent and choiceBranchRef are provided
on the same choice. However, it is not an error if a discriminator exists on a choice branch along
with an elementID.

15.1.3 Unparsing Choices

On unparsing there is the question of how one identifies the appropriate schema choice branch
corresponding to the data in the infoset. This is complicated by the fact that the children may not
be elements. They may themselves be sequences or choices.The selection of the choice branch
is as follows: 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 selected and if subsequently a processing error occurs, this
selection is not revisited (that is, there is no backtracking).

To avoid any unintended behavior, all the children of a choice can be modeled as elements.

.

Comment [m509]: But what is the behavior in
this case?

SMH: We agreed that any discriminator
resolves the nearest active PoU, and that still
applied here – so if choiceBranchRef was also
present, the discriminator would resolve a
higher PoU – so use with care.

Resolved that you can have a discriminator with
either initiatedContent or with direct dispatch,
but you can't have both initiatedContent AND
direct dispatch.

DONE

Comment [m510]: Errata 3.15 Done

Comment [m511]: Errata 2.81
DONE

16. Properties for Array Elements and Optional Elements

These properties are for array elements (XSDL maxOccurs >1 or unbounded) or optional elements (XSDL minOccurs = 0 and maxOccurs = 1) and apply to local element declarations and element references. The properties handle a logical one-dimensional array of any simple or complex type.

Property Name	Description
occursCountKind	Enum
	Specifies how the actual number of occurrences is to be established.
	Valid values 'fixed', 'expression', 'parsed', 'implicit' ,'stopValue'.
	'fixed' means use the XSDL maxOccurs property.
	'expression' means use the dfdl:occursCount property.
	'parsed' means that the number of occurrences is determined solely by speculative parsing.
	'implicit' means that the number of occurrences is determined by speculative parsing in conjunction with the XSDL minOccurs and maxOccurs properties.
	'stopValue' means look for a mandatory logical stop value which signifies the end of the occurrences.
	These values are described in detail in section 16.1.
	Annotation: dfdl:element
occursCount	DFDL Expression
	Specifies the number of occurrences of the element.
	This property is computed by way of an expression which returns a non-negative integer. The expression must not contain forward references to elements which have not yet been processed.
	Annotation: dfdl:element,
occursStopValue	List of DFDL Logical Values
	A space separated list of logical values that specify the alternative logical stop values for the element.
	Required only when dfdl:occursCountKind='stopValue'.
	When parsing then if an occurrence of the element has a logical value that matches one of the values in this list then the parser must not expect any more occurrences of the element.
	On unparsing the first value will be inserted as an additional final occurrence in the array after all of the occurrences in the infoset have been output.
	Annotation: dfdl:element

When XSDL minOccurs = 1 and maxOccurs = 1, the above properties are not used, and the behavior is as if dfdl:occursCountKind was 'fixed' as described in section 16.1.1.

Comment [SMH512]: Action 140 folded in throughout this section.

TODO IN PROGESS

Comment [SMH513]: Errata 3.11 adds new enum 'implicit'

DONE

Comment [SMH514]: Errata 2.31 DONE

Comment [SMH515]: Errata 2.36 changes this so only an expression is allowed. **DONE**

16.1   The dfdl:occursCountKind property

16.1.1   dfdl:occursCountKind 'fixed'

The enum 'fixed' should be used when the number of occurrences is always the same. The number is provided by the XSDL maxOccurs property.

When parsing, maxOccurs occurrences are expected in the data. It is a processing error if less than maxOccurs occurrences are found or defaulted. The parser stops looking for occurrences when maxOccurs have been found or defaulted.

When unparsing, maxOccurs occurrences are expected in the infoset. It is a processing error if less than maxOccurs occurrences are found or defaulted, or if more than maxOccurs occurrences are found.

It is a schema definition error if minOccurs is not equal to maxOccurs.

16.1.2   dfdl:occursCountKind 'implicit'

The enum 'implicit' should be used when the number of occurrences is to be established using speculative parsing, and there are lower and upper bounds to control the speculation. The bounds are provided by the XSDL minOccurs and XSDL maxOccurs properties.

When parsing, up to maxOccurs occurrences are expected in the data. It is a processing error if less than minOccurs occurrences are found or defaulted. The parser stops looking for occurrences when either minOccurs have been found or defaulted and speculative parsing does not find another occurrence, or maxOccurs have been found or defaulted.

When unparsing, up to maxOccurs occurrences are expected in the infoset. It is a processing error if less than minOccurs occurrences are found or defaulted, or if more than maxOccurs occurrences are found.

16.1.3   dfdl:occursCountKind 'parsed'

The enum 'parsed' should be used when the number of occurrences is to be established solely using speculative parsing.

When parsing, any number of occurrences is expected in the data. The parser stops looking for occurrences when speculative parsing does not find another occurrence. If validation is enabled, it is a validation error if less than XSDL minOccurs occurrences are found or defaulted, or greater than XSDL maxOccurs occurrences are found.

When unparsing, any number of occurrences is expected in the infoset. If validation is enabled, it is a validation error if less than minOccurs occurrences are found or defaulted, or if more than maxOccurs occurrences are found.

16.1.4   dfdl:occursCountKind 'expression'

The enum 'expression' should be used when the number of occurrences is calculated by evaluating a DFDL expression.

When parsing, the dfdl:occursCount expression is evaluated and provides the number of occurrences expected in the data. It is a processing error if less than dfdl:occursCount occurrences are found or defaulted. The parser stops looking for occurrences when dfdl:occursCount occurrences have been found. If validation is enabled, it is a validation error if less than XSDL minOccurs occurrences are found or defaulted, or more than XSDL maxOccurs occurrences are found.

When unparsing, the behavior is the same as for 'parsed'.

It is a schema definition error if dfdl:occursCount is not provided or in scope.

16.1.5   dfdl:occursCountKind 'stopValue'

The enum 'stopValue' should be used when the the end of the array is signaled by an occurrence having a logical value that is equal to one of the specified 'stop values'.

When parsing, any number of occurrences is expected in the data, followed by an occurrence which is a stop value as specified by dfdl:occursStopValue. It is a processing error if a stop value occurrence is not found in the data (including when there are zero other occurrences). The parser stops looking for occurrences once a stop value has been found. If validation is enabled, it is a validation error if less than XSDL minOccurs occurrences are found or defaulted, or more than XSDL maxOccurs occurrences are found, not including the stop value.

When unparsing, the behavior is the same as for 'parsed', with the addition that a stop value occurrence is output after the last infoset occurrence. If dfdl:occursStopValue provides multiple stop values then the first is used.

The stop value itself is **not** added to the infoset when parsing, nor should a stop value be placed in the infoset for unparsing.

It is a schema definition error if dfdl:occursStopValue is not provided or in scope.

It is a schema definition error if the type of the element is complex.

It is a schema definition error if any of the stop values provided by dfdl:occursStopValue do not conform to the simple type of the element.

16.2   Default Values for Arrays

When parsing, required occurrences that have empty representation may trigger the application of a default value, as described in Section 9.4.2 Defaults When Parsing.

When unparsing, required occurrences that are missing from the infoset may trigger the application of a default value, as described in Section 9.4.3 Element Defaults When Unparsing.

The application of default values is **not** dependent on dfdl:occursCountKind, only on whether the occurrence is required or optional, whether there is a default value specified, and whether the data contains the empty representation. Section 9.4 contains the details.

16.3   Arrays with DFDL Expressions

If the value of a DFDL property of an array element (other than dfdl:occursCount) is given by a DFDL Expression, then the expression must be re-evaluated for each occurrence of the element in case the value changes.

16.4   Points of Uncertainty

When parsing an array, points of uncertainty (PoU) only occur for certain values of occursCountKind, as follows:

- fixed. No PoU (maxOccurs occurrences expected).
- implicit. PoU exists after minOccurs occurrences found and until maxOccurs found.
- parsed. PoU exists for all occurrences
- expression. No PoU (occursCount occurrences expected)
- stopValue. No PoU (stopValue must always be present, even when minOccurs=0).

16.5   Arrays and Sequences

In some situations arrays of elements and sequence groups of elements seem to be similar; however, there is no notion of the array itself independent of its contained elements. Arrays are distinctly different from sequence groups in this way.

A sequence can have its own initiator, and a complex element having that sequence as its content can also have its own initiator, so you could express two different initiators.

Unlike a sequence group, an array does not have its own initiator, terminator, or alignment. Those properties apply to each element occurrence of the array. To give an alignment, initiator,

Comment [SMH516]: Errata 2.31 : **DONE**

Comment [SMH517]: TODO: Move to section 9.1.1 which is where PoUs are discussed.

separator or terminator to an entire array you must enclose the element declaration for the array in a sequence group and specify the alignment, separator, initiator and terminator on the sequence group.

16.6   Forward Progress Requirement

It is a processing error when an array is potentially unbounded and the position in the data does not move during the parsing of an occurrence of the element including any associated separator. This is to prevent an infinite loop. An array is potentially unbounded if any of the following are true:

- dfdl:occursCountKind is 'parsed' or 'stopValue'
- dfdl:occursCountKind is 'implicit' and XSDL maxOccurs is 'unbounded'

16.7   Parsing Occurrences with Non-Normal Representation

When parsing a single array, it is possible to extract occurrences that have different representations (nil, empty, normal, absent) although with some values of dfdl:lengthKind certain combinations of representations are not possible.

Occurrences with nil representation are added to the infoset with value 'nil'.

Occurrences with empty representation may or may not be added to the infoset, as described in Section 9.4. If a required occurrence is not added to the infoset, it may be a processing error, dependent on dfdl:occursCountKind as described in section 16.1.

Occurrences with absent representation are not added to the infoset. For a required occurrence it may be a processing error, dependent on dfdl:occursCountKind as described in section 16.1.

16.8   Sparse Arrays

Consider parsing an array where optional occurrences with empty representation are present in the data, but there are also later optional occurrences present with normal representation. Such an array is sometimes called a 'sparse array'.

- If the indices of the occurrences are significant and need to be preserved, then the array may be modelled using an element with XSDL nillable 'true', dfdl:nilKind 'literalValue', dfdl:nilValue '%ES;' and dfdl:nilValueDelimiterPolicy the same as dfdl:emptyValueDelimiterPolicy. The occurrences with empty representation now become occurrences with nil representation, and will produce nil values in the infoset, so the absolute positions of all occurrences are preserved.

- If the indices of the occurrences are not significant, then the array should be modelled using an element with XSDL nillable 'false'. Optional occurrences with empty representation will not create items in the infoset, so the absolute position of any optional occurrences with normal representation is not preserved. Optional occurrences with empty representation are therefore skipped.

17. Calculated Value Properties.

This section describes properties which allow the creation of calculated elements. When parsing, the value of a calculated element is derived using a DFDL Expression, and not by processing bytes from the data stream. When unparsing, the value of a calculated element is derived using a DFDL Expression, and is not obtained from the infoset in the usual way.

Calculated elements allow a technique that is commonly called layering. In this technique, some elements are said to be in the physical layer, and some in the logical layer. When parsing, the logical layer values are computed from physical layer values. When unparsing the opposite occurs, that is the physical layer values are computed from the logical layer values.

Calculated elements are commonly used with hidden elements so as to hide the physical layer elements so that they do not become part of the infoset.

When a DFDL Schema is used to both parse and unparse data, then a calculated element on parsing will normally have one or more calculated elements on unparsing.

These properties apply to elements of simple type.

Property Name	Description
inputValueCalc	DFDL Expression
	An expression that calculates the value of the element when parsing.
	The element having the inputValueCalc property is called a derived element, and the elements referenced from the inputValueCalc expression are called representation elements.
	It is a schema definition error if the result type of the expression does not conform to the base type of the element.
	The value created using inputValueCalc is validated like any other element value (when validation is enabled).
	An element that specifies an inputValueCalc expression has no representation of its own in the data stream. All other DFDL representation properties are ignored
	The element must not be optional nor an array nor be global.
	The DFDL Expression must not refer to this element nor cause a circular reference to this element. The expression must not contain forward references to elements which have not yet been processed.
	It is a schema definition error if dfdl:inputValueCalc is specified on an element which has an XSDL fixed or default property.
	It is a schema definition error if dfdl:inputValueCalc and dfdl:outputValueCalc are specified on the same element.
	It is not possible to place this property in scope on a dfdl:format annotation.
	Annotation: dfdl:element
outputValueCalc	DFDL Expression
	An expression that calculates the value of the current element when unparsing.
	The element must not be optional nor an array nor be global.
	It is a schema definition error if the result type of the expression does

Comment [SMH518]: Errata 2.7. **DONE**

Comment [SMH519]: Errata 2.35 expands this

DONE. Note Schema Definition error for type-error, not processing error.

Comment [SMH520]: Errata 3.2 DONE

Comment [SMH521]: Need to add that empty string is a schema definition error.

Comment [m522]: Not sure. The type is DFDL Expression. So it' can't be "" because that's not a DFDL expression.

The runtime value *can* be a empty string because the type of the element might be xs:string.

In general, other places specify that a DFDL Expression cannot be "". You must have braces, and you must have something inside the braces that is a valid expression. I don't think we should detail that here redundantly.

DONE

Comment [SMH523]: Errata 3.2 changes property to behave like dfdl:ref

DONE

Comment [SMH524]: Need to make same change here as for inputValueCalc above.

Comment [m525]: Fixed by deleting misleading sentence.
DONE

Comment [SMH526]: Errata 2.35 expands this

DONE

Comment [SMH527]: Errata 3.2 **DONE**

not conform to the base type of the element.

The value created using outputValueCalc is validated like any other element value (when validation is enabled).

The value for the element, if any, in the infoset is ignored.

The DFDL expression must not refer to this element nor cause a circular reference to this element. The expression may contain forward references to elements which have not yet been processed.

It is a schema definition error if dfdl:outputValueCalc is specified on an element which has an XSDL fixed or default property.

It is a schema definition error if dfdl:inputValueCalc and dfdl:outputValueCalc are specified on the same element.

It is not possible to place this property in scope on a dfdl:format annotation.

Annotation: dfdl:element

Comment [m528]: (OPEN)
Do we want to say something about implementations placing limits here?

Can that be said centrally elseswhere, e.g., in the section about the expression language?

OPEN

Comment [SMH529]: Need to add that empty string is a schema definition error.

Don't think so - See comment above on inputValueCalc.

DONE

Comment [SMH530]: Errata 3.2 changes property to behave like dfdl:ref

DONE

17.1   Example: 2d Nested Array

Consider this simple example. The data stream contains two elements giving the number of rows and number of columns of an array of numbers. The representation of the array is stored after these two elements.

```
<xs:complexType name="array">
  <xs:sequence dfdl:initiator="" >

    <xs:sequence dfdl:hiddenGroupRef="tns:hiddenArrayCounts"/>

    <xs:element name="rows" maxOccurs="unbounded"
               dfdl:occursCountKind="expression"
               dfdl:occursCount="{ ../nrows }">
      <xs:complexType>
        <xs:sequence>
          <xs:element name="cols" type="xs:float" maxOccurs="unbounded"
                     dfdl:occursCountKind="expression"
                     dfdl:occursCount=" { ../../ncols } " />
        </xs:sequence>
      </xs:complexType>
    </xs:element>
  </xs:sequence>
</xs:complexType>

<xs:group name="hiddenArrayCounts" >
  <xs:sequence>
    <xs:element name="nrows" type="xs:unsignedInt"
               dfdl:representation="binary"
               dfdl:lengthKind="implicit"
               dfdl:outputValueCalc="{ count(../rows) }"/>
    <xs:element name="ncols" type="xs:unsignedInt"
               dfdl:representation="binary"
               dfdl:lengthKind="implicit"
               dfdl:outputValueCalc=
               "{  if ( count(../rows) ge 1 )
                   then
                       count(../rows[1]/cols)
```

```
                            else
                               0
                          }"/>
     </xs:sequence>
</xs:group>
```

In the example above we see that there are two hidden elements named 'nrows' and 'ncols'. These hidden elements' values are computed when unparsing from the number of occurrences in the 'rows' and 'cols' repeating elements. The 'rows' and 'cols' repeating elements number of occurrences are computed when parsing from the hidden elements 'nrows' and 'ncols'.

17.2 Example: Three-Byte Date

Logically, the data is a date.

```
<xs:element name="d" type="date"/>
```

Physically, it is stored as 3 single byte integers.

The format of this data is expressed as this schema:

```
<xs:sequence dfdl:representation="binary">
                <xs:element name="mm" type="byte" />
                <xs:element name="dd" type="byte" />
                <xs:element name="yy" type="byte"/>
</xs:sequence>
```

This physical representation can be hidden so that it does not become part of the infoset:

```
<xs:sequence>
  <xs:sequence dfdl:hiddenGroupRef="tns:hiddenpDate"/>

  <xs:element name="d" type="date">
    …
  </xs:element>
</xs:sequence>

<xs:group name="hiddenpDate" >
  <xs:sequence>
    <xs:element name="pdate">
      <xs:complexType>
        <xs:sequence dfdl:representation="binary">
          <xs:element name="mm" type="byte" />
          <xs:element name="dd" type="byte" />
          <xs:element name="yy" type="byte"/>
        </xs:sequence>
      </xs:complexType>
    </xs:element>
  </xs:sequence>
</xs:group>
```

A calculation can be used to compute the logical date element 'd' from the physical 'pdate' when parsing:

```
<xs:sequence>
  ... hidden pdate here ...

  <xs:element name="d" type="date">
    <xs:annotation><xs:appinfo source="http://www.ogf.org/dfdl/">
      <dfdl:element>
        <dfdl:property name="inputValueCalc">
          {
            fn:date(fn:concat(if(../pdate/yy gt 50 )then "19" else "20",
```

```
                                              if ( ../pdate/yy gt 9 )
                                                  then fn:string(../pdate/yy)
                                                  else fn:concat("0",
                                                          fn:string(../pdate/yy)),
                                                      "-",
                                                      fn:string(../pdate/mm),
                                                      "-",
                                                      fn:string(../pdate/dd)))
              }
          </dfdl:property>
        </dfdl:element>
      </xs:appinfo></xs:annotation>
  </xs:element>
  ...
</xs:sequence>
```

The expression above assembles a string resembling, for example, "2005-12-17" or "1957-3-9" which is the string representation of a date that is acceptable to the fn:date constructor function. The hidden element 'pdate' is referenced by relative paths. The expression '../pdate/yy' accesses an element of type 'int', and the fn:string constructor function turns it into an integer.

Finally, we must handle the unparse case where the physical layer is computed from the logical layer:

```
<xs:sequence dfdl:representation="binary"
  <xs:element name="mm" type="byte"
      dfdl:outputValueCalc="{ fn:month-from-date(../d) }" />
  <xs:element name="dd" type="byte"
      dfdl:outputValueCalc="{ fn:day-from-date(../d) }" />
  <xs:element name="yy" type="byte"
      dfdl:outputValueCalc="{ fn:year-from-date(../d) idivmod 100 }"/>
</xs:sequence>
```

The entire example in one place:

```
<xs:sequence>
  <xs:sequence dfdl:hiddenGroupRef="tns:hiddenpDate"/>

  <xs:element name="d" type="date">
    <xs:annotation><xs:appinfo source="http://www.ogf.org/dfdl/">
      <dfdl:element>
        <dfdl:property name="inputValueCalc">
          {
           fn:date(fn:concat(if(../pdate/yy gt 50) then "19" else "20",
                             if ( ../pdate/yy gt 9 )
                                 then fn:string(../pdate/yy)
                                 else fn:concat("0",
                                         fn:string(../pdate/yy)),
                                     "-",
                                     fn:string(../pdate/mm),
                                     "-",
                                     fn:string(../pdate/dd)))
          }
        </dfdl:property>
      </dfdl:element>
    </xs:appinfo></xs:annotation>
  </xs:element>
  ...
</xs:sequence>
```

```
<xs:group name="hiddenpDate" >
  <xs:sequence>
    <xs:element name="pdate">
      <xs:complexType>
        <xs:sequence dfdl:representation="binary">
          <xs:element name="mm" type="byte"
                 dfdl:outputValueCalc="{ fn:month-from-date(../d) }" />
          <xs:element name="dd" type="byte"
                 dfdl:outputValueCalc="{ fn:day-from-date(../d) }" />
          <xs:element name="yy" type="byte"
                 dfdl:outputValueCalc="{ fn:year-from-date(../d)
idivmod 100 }" />
        </xs:sequence>
      </xs:complexType>
    </xs:element>
  </xs:sequence>
</xs:group>
```

The above sequence contains logically only a single date element.

18. External Control of the DFDL Processor

In addition to providing the DFDL schema and data to be parsed or serialized, DFDL Schemas can also be parameterized by external variables,.

DFDL processors can provide means to specify:

1. The data to be processed: a data stream when parsing or an infoset when unparsing.

2. The DFDL schema to be used

3. The *distinguished root node* element declaration to be used (specifying both name of element and namespace of that name)

4. Values for external variables

Notice also that like any XML schema a DFDL schema can have multiple top-level element declarations, so the distinguished root node is necessary to indicate which of these top-level element declarations is to be the starting point for processing data. The distinguished root node may be omitted if the DFDL schema contains only one top-level element declaration.

The mechanism by which a DFDL processor is controlled is not specified by this standard. For example, command line DFDL processors may use command line options, but DFDL processors embedded in other kinds of software systems may need other mechanisms.

19. Built-in Specifications

For convenience, a standard set of named DFDL format definitions may be provided with DFDL processors. These built-in format definitions may be imported by DFDL schema authors.

20. Conformance

DFDL conformance can be claimed for schema documents and for processors

A schema document conforms to this specification if it conforms to the subset of XML Schema 1.0 defined in section 5.1 DFDL Subset of XML Schema and consists of components which individually and collectively satisfy all the relevant constraints specified in this document.

Conformance may be claimed separately for a DFDL parser, a DFDL unparser or a DFDL processor that parses and unparses.

1. A DFDL processor claiming conformance MUST identify the level of conformance and version specification claimed.

2. A minimal conforming DFDL processor conforms to this specification when it implements all the non-optional features defined in this document.

3. An extended conforming DFDL processor conforms to the specification when it implements all the non-optional features and some of the optional features defined in this document.

4. A fully conforming DFDL processor conforms to the specification when it implements all the features defined in this document.

See Section 21 Optional DFDL Features for the list of optional feature

It is the intention of the DFDL Work Group to provide a conformance test suit to help verify conformance with this specification..

21. Optional DFDL Features

The following table lists the features of the DFDL language that are considered optional for DFDL processor implementations. This list admits very small subsets of the full DFDL specification. For example, a binary-only subset without any expressions or variables is specifically allowed.

Feature	Detection
Validation	External switch
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"
Simple type restrictions	xs:simpleType in xsd
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"
Nils	XSDL nillable='true' in xsd
Defaults	XSDL default or fixed in xsd
Bi-Directional text.	dfdl:textBiDi='yes'
Lengths in Bits	dfdl:alignmentUnits='bits' or dfdl:lengthUnits='bits'
Delimited lengths and representation binary element	dfdl:representation='binary' (or implied binary) and dfdl:lengthKind='delimited'
Regular expressions	dfdl:lengthKind='pattern', dfdl:assert with dfdl:testkind 'pattern' , dfdl:discriminator with dfdl:testkind 'pattern'
Zoned numbers	dfdl:textNumberRep='zoned'
IBM 390 packed numbers	dfdl:binaryNumberRep='packed'
IBM 390 packed calendars	dfdl:binaryCalendarRep='packed'
IBM 390 floats	dfdl:binaryFloatRep='ibm390Hex'
Unordered sequences	dfdl:sequenceKind='unordered'
Floating elements	dfdl:floating='yes'
dfdl functions in expression language	dfdl:functions in expression
Hidden groups	dfdl:hiddenGroupRef <> "
Calculated values	dfdl:inputValueCalc <> " or dfdl:outputValueCalc <> "
Escape schemes	dfd:defineEscapeScheme in xsd
Extended encodings	Any dfdl:encoding value beyond the core list
Asserts	dfdl:assert in xsd
Discriminators	dfdl:discriminator in xsd

Comment [m531]: Errata 3.17 done.

Prefixed lengths	dfdl:lengthKind='prefixed'
Variables	dfdl:defineVariable,
	dfdl:newVariableInstances,
	dfdl:setVariable
	Variables in DFDL expression language
	Note that variables as a feature is dependent on the Expressions feature.
BCD calendars	dfdl:binaryCalendarRep="bcd"
BCD numbers	dfdl:binaryNumberRep="bcd"
Multiple schemas	xs:include or xs:import in xsd
IBM 4690 packed numbers	dfdl:binaryNumberRep="ibm4690Packed"
IBM 4690 packed calendars	dfdl:binaryCalendarRep="ibm4690Packed"

Table 23 Optional DFDL features

In order to provide portability of a DFDL schema, a minimal or extended conforming processor must issue warnings about any DFDL properties it does not implement. This warning can simply state that the property was not recognized.

(This allows the implementation to simply have no knowledge of properties it does not need for the subset of features it implements.)

For example if the bi-directional text feature is not implemented, then the implementation will most likely not recognize the dfdl:textBiDi property at all. Such an implementation must issue a warning that the 'dfdl:textBiDi' property was not recognized.

It is a schema definition error if a DFDL schema uses an optional feature that is not supported by a minimal or extended conforming processor.
.

Comment [SMH532]: Really? There is still value in providing expressions without variables (IBM DFDL does this today).

Removed misleading sentence

DONE

Comment [m533]: I'm pretty sure we missed this.

SMH: I think the assumption was that BCD was supported by the core.

DONE (BCD is optional)

Comment [SMH534]: Errata 3.8 relaxes this.

Comment [m535]: DONE

I wasn't sure what to change this to. I am concerned about schemas being used by a subset processor, which ignore some of the properties on the schema, but don't get an SDE, but then attempt to parse data, and fail.... but they're ignoring properties (intentionally) that need to be used to successfully parse data.

Perhaps any conforming processor must recognize all DFDL v1.0 properties (just the property names, not validating the values) and when found output a warning (via unspecified mechanism) indicating that the property is not implemented.

SMH: The latter was my original intent, but we agreed on a WG call to relax this, see errata 3.8 which says a DFDL processor "…that does not implement some optional features does not have to validate properties required by those optional features, though it is recommended that it does so for portability reasons". Happy to revisit.

Comment [m536]:

Well if a schema uses an optional feature that is not supported, it really wants to be an SDE.

OTOH we don't want to add a burden to implementors who want to create minimal implementations.

So a middle ground is to require an SDE, but for any property the implementation does not use. That's easier than requiring the implementation to recognize and check things like textBiDi="no". Instead one would get "unrecognized property textBiDi with value 'no'.

An implementation just has to pick off all attributres on dfdl annotation elements, and all short-form attributes in the dfdl namespace, and all dfdl:property elements with unknown names, and issue an "unrecognized property" SDE. So, this doesn't seem so bad to me.

DONE (resolved – must warn about properties they don't recognize)

22. Property Precedence

22.1 Parsing

The following list gives the order in which DFDL properties are examined when the DFDL parser is positioned at a particular component in the DFDL schema, and about to parse the bitstream modeled by that component.

22.1.1 dfdl:element (simple) and dfdl:simpleType

- *Parsing: calculated value (does not apply to dfdl:simpleType or to global elements)*
    - o dfdl:inputValueCalc
- *Parsing: common*
    - o dfdl:encoding
        - ▪ 'UTF-16' 'UTF-16BE' 'UTF-16LE'
            - ▪ dfdl:utf16Width
    - o dfdl:ignoreCase
- *Parsing: nillable*
    - o XSDL nillable *(does not apply to dfdl:simpleType)*
        - ▪ dfdl:nilKind
            - ▪ *"literalValue", "logicalValue", "literalCharacter"*
                - ▪ dfdl:nilValue
- *Parsing: occurrences (does not apply to dfdl:simpleType or to global elements)*
    - o dfdl:floating
    - o (maxOccurs > 1 or unbounded) or (minOccurs = 0 and maxOccurs = 1)
        - ▪ dfdl:occursCountKind
            - ▪ *"expression"*
                - ▪ dfdl:occursCount
            - ▪ *"fixed", "implicit"*
                - ▪ minOccurs
                - ▪ maxOccurs
            - ▪ *"parsed"*
            - ▪ *"stopValue"*
                - ▪ dfdl:occursStopValue
- *Parsing: identification, framing & extraction*
    - o dfdl:leadingSkip
        - ▪ dfdl:alignmentUnits
    - o dfdl:alignment
        - ▪ dfdl:alignmentUnits
    - o dfdl:initiator
        - ▪ dfdl:nilValueDelimiterPolicy *(does not apply to dfdl:simpleType)*

Comment [m537]: (OPEN)

This section has some of the issues the grammar section has.

That is, that it's starting to be recognizably flawed and doesn't match what implementations really have to do.

Comment [m538]: Do we have to fix this, or has this precedence outline now become obsolete as a concept?

E.g., if I recurse through the grammar, the properties that apply to each region define exactly this precedence order don't they?

If we used the grammar as an outline. Make 2 copies (one for parsing, one for unparsing), then annotated each outline node with the properties that are referenced first at that node, do we get the same thing this is trying to be?

Comment [SMH539]: Errata 3.3 & 2.7. **DONE**

Comment [SMH540]: Now that byte order is only applicable to binary reps, it could be moved down

Comment [m541]: Moved. DONE

Comment [SMH542]: Errata 2.8 & 2.58 mean these are examined earlier

DONE: moved before occurrances.

Comment [SMH543]: Is just moving before occurrences sufficient? That on its own is not sufficient to reflect that nilLiteralCharacter processing takes place before trimmimng whereas the others are after, for example.

OPEN – see overall comment above.

Comment [SMH544]: Add implicit DONE

- dfdl:emptyValueDelimiterPolicy
- ○ dfdl:representation *"text" or xs:simpleType is 'string'*
  - dfdl:lengthKind
    - *"implicit"*
      - xs:maxLength *or* dfdl:textBooleanTrueRep/dfdl:textBooleanFalseRep
      - dfdl:lengthUnits
    - *"explicit"*
      - dfdl:length
      - dfdl:lengthUnits
    - *"prefixed"*
      - dfdl:prefixLengthType
      - dfdl:prefixIncludesPrefixLength
      - dfdl:lengthUnits
    - *"pattern"*
      - dfdl:lengthPattern
    - *"delimited", "endOfParent"*
      - *None*
  - dfdl:textTrimKind
    - dfdl:textStringPadCharacter, dfdl:textNumberPadCharacter, dfdl:textBooleanPadCharacter or dfdl:textCalendarPadCharacter
    - dfdl:textStringJustification, dfdl:textNumberJustification, dfdl:textBooleanJustification or dfdl:textCalendarJustification
  - dfdl:escapeSchemeRef
  - dfdl:textBidi
    - dfdl:textBidiOrdering
    - dfdl:textBidiOrientation
- ○ dfdl:representation *"binary" or xs:simpleType is 'hexBinary'*
  - dfdl:lengthKind
    - *"implicit"*
      - xs:maxLength *or* xs:simpleType
      - dfdl:lengthUnits
    - *"explicit"*
      - dfdl:length
      - dfdl:lengthUnits
    - *"prefixed"*
      - dfdl:prefixLengthType
      - dfdl:prefixIncludesPrefixLength

Comment [SMH545]: Errata 2.60 changes name **DONE**

Comment [SMH546]: Is textBidiSymmetric missing from here or later (for string)?

Comment [m547]: (OPEN) - textBiDi stuff.

Comment [m548]: Theoretically, byteOrder could go even further down, but gets snarled up in e.g., that prefixed might be a multi-byte integer as the prefix value

SMH: I think it needs pushing down into the conversion section

DONE

- ▪ dfdl:lengthUnits
  - ▪ *"delimited", "endOfParent"*
    - ▪ *None*
  - o dfdl:terminator
    - ▪ dfdl:nilValueDelimiterPolicy *(does not apply to dfdl:simpleType)*
    - ▪ dfdl:emptyValueDelimiterPolicy
    - ▪ dfdl:documentFinalTerminatorCanBeMissing
  - o dfdl:trailingSkip
    - ▪ dfdl:alignmentUnits
- • *Parsing: conversion*
  - o xs:type
    - ▪ *"Number"*
      - ▪ dfdl: decimalSigned
      - ▪ dfdl:representation
        - ▪ *"text"*
          - ▪ dfdl:textNumberRep
            - ▪ *"standard"*
              - ▪ dfdl:textNumberPattern
              - ▪ dfdl:textStandardDecimalSeparator
              - ▪ dfdl:textStandardGroupingSeparator
              - ▪ dfdl:textStandardExponentRep
              - ▪ dfdl:textNumberCheckPolicy
              - ▪ dfdl:textStandardInfinityRep
              - ▪ dfdl:textStandardNaNRep
              - ▪ dfdl:textNumberRounding
                - ▪ *"explicit"*
                  - ▪ dfdl:textNumberRoundingMode
                  - ▪ dfdl:textNumberRoundingIncrement
              - ▪ dfdl:textStandardZeroRep
              - ▪ dfdl:textStandardBase
            - ▪ *"zoned"*
              - ▪ *d*fdl:textNumberPattern
              - ▪ dfdl:textNumberCheckPolicy
              - ▪ dfdl:textNumberRounding
                - ▪ *"explicit"*

Comment [SMH549]: This needs removing unless we decide to allow pattern for packed or bcd reps

REMOVED "pattern" since not allowed for binary.
DONE

Comment [SMH550]: Errata 2.4. **DONE**

Comment [SMH551]: Errata 2.8 & 2.58 mean these are examined earlier
DONE (moved earlier)

Comment [SMH552]: Errata 2.46 changes name .**DONE**

Comment [SMH553]: NaN
DONE

- ▪ dfdl:textNumberRoundingMode
                        - ▪ dfdl:textNumberRoundingIncrement
                - ▪ dfdl:textZonedSignStyle
        - ▪ *"binary"*
            - ▪ dfdl:byteOrder
            - ▪ *xs:decimal and restrictions*
                - ▪ dfdl:binaryNumberRep
                    - ▪ *"packed"*
                        - ▪ dfdl:binaryPackedSignCodes
                        - ▪ dfdl:binaryDecimalVirtualPoint
                        - ▪ dfdl:binaryNumberCheckPolicy
                    - ▪ *"bcd", "ibm4690Packed"*
                        - ▪ dfdl:binaryDecimalVirtualPoint
                        - ▪ dfdl:binaryNumberCheckPolicy
                    - ▪ *"binary"*
                        - ▪ dfdl:binaryDecimalVirtualPoint
            - ▪ *xs:float, xs:double*
                - ▪ dfdl:binaryFloatRep
- ▪ *"String"*
- ▪ *"Calendar"*
    - ▪ dfdl:representation
        - ▪ *"text"*
            - ▪ dfdl:calendarPatternKind "*explicit*"
                - ▪ dfdl:calendarPattern
            - ▪ dfdl:calendarCheckPolicy
            - ▪ dfdl:calendarTimeZone
            - ▪ dfdl:calendarObserveDST
            - ▪ dfdl:calendarFirstDayOfWeek
            - ▪ dfdl:calendarDaysInFirstWeek
            - ▪ dfdl:calendarCenturyStart
            - ▪ dfdl:calendarLanguage
        - ▪ *"binary"*

- ▪ dfdl:byteOrder
- ▪ dfdl:binaryCalendarRep
  - ▪ *"packed"*
    - ▪ dfdl:packedDecimalSignCodes
    - ▪ dfdl:binaryNumberCheckPolicy
    - ▪ dfdl:calendarPatternKind
      - ▪ "*explicit*"
        - ▪ dfdl:calendarPattern
    - ▪ dfdl:calendarCheckPolicy
    - ▪ dfdl:calendarTimeZone
    - ▪ dfdl:calendarObserveDST
    - ▪ dfdl:calendarFirstDayOfWeek
    - ▪ dfdl:calendarDaysInFirstWeek
    - ▪ dfdl:calendarCenturyStart
  - ▪ *"bcd", "ibm4690Packed"*
    - ▪ dfdl:binaryNumberCheckPolicy
    - ▪ dfdl:calendarPatternKind
      - ▪ "*explicit*"
        - ▪ dfdl:calendarPattern
    - ▪ dfdl:calendarCheckPolicy
    - ▪ dfdl:calendarTimeZone
    - ▪ dfdl:calendarObserveDST
    - ▪ dfdl:calendarFirstDayOfWeek
    - ▪ dfdl:calendarDaysInFirstWeek
    - ▪ dfdl:calendarCenturyStart
  - ▪ *"binarySeconds", "binaryMilliseconds"*
    - ▪ dfdl:binaryCalendarEpoch
- ▪ *"Opaque"*
- ▪ *"Boolean"*
  - ▪ dfdl:representation
    - ▪ *"text"*
      - ▪ dfdl:textBooleanTrueRep
      - ▪ dfdl:textBooleanFalseRep
    - ▪ *"binary"*
      - ▪ dfdl:byteOrder
      - ▪ dfdl:binaryBooleanTrueRep
      - ▪ dfdl:binaryBooleanFalseRep
- o dfdl:useNilForDefault *(does not apply to dfdl:simpleType)*

> **Comment [SMH554]:** No longer used for packed
>
> REMOVED.
> DONE

> **Comment [SMH555]:** No longer used for bcd
>
> REMOVED
> DONE

- ▪ *"true"*
  - ▪ *None*
- ▪ *"false"*
  - ▪ XSDL default or:fixed

22.1.2 dfdl:element (complex)

- *Parsing: common*
  - o dfdl:encoding
    - ▪ 'UTF-16' 'UTF-16BE' 'UTF-16LE'
      - ▪ dfdl:utf16Width
  - o dfdl:ignoreCase

> **Comment [SMH556]:** Remove byteOrder
> DONE

- *Parsing: nillable*
  - o XSDL nillable
    - ▪ dfdl:nilKind
      - ▪ *"literalValue"*
        - ▪ dfdl:nilValue (must be "%ES;")
- *Parsing: occurrences (does not apply to global elements)*
  - o dfdl:floating
  - o (maxOccurs > 1 or unbounded) or (minOccurs = 0 and maxOccurs = 1)
    - ▪ dfdl:occursCountKind

> **Comment [SMH557]:** Add implicit
> DONE

- - - - - ▪ *"expression"*
        - ▪ dfdl:occursCount
      - ▪ *"fixed"* , "*implicit*"
        - ▪ minOccurs
        - ▪ maxOccurs
      - ▪ *"parsed"*
- *Parsing: identification, framing & extraction*

> **Comment [SMH558]:** Errata 3.10 adds nillable complex elements
> DONE

- - o dfdl:leadingSkip
    - ▪ dfdl:alignmentUnits
  - o dfdl:alignment
    - ▪ not *"implicit"*
      - ▪ dfdl:alignmentUnits
  - o dfdl:initiator
    - ▪ dfdl:nilValueDelimiterPolicy
    - ▪ dfdl:emptyValueDelimiterPolicy
  - o dfdl:lengthKind
    - ▪ *"explicit"*
      - ▪ dfdl:length

- - - ▪ dfdl:lengthUnits
- - ▪ *"prefixed"*
- - - ▪ dfdl:prefixLengthType
- - - ▪ dfdl:prefixIncludesPrefixLength
- - - ▪ dfdl:lengthUnits
- - ▪ *"pattern"*
- - - ▪ dfdl:lengthPattern
- - ▪ *"implicit", "delimited", "endOfParent"*

Comment [SMH559]: Errata 2.5. **DONE**

- - - ▪ *None*
- o dfdl:terminator
- - ▪ dfdl:nilValueDelimiterPolicy
- - ▪ dfdl:emptyValueDelimiterPolicy
- - ▪ dfdl:documentFinalTerminatorCanBeMissing
- o dfdl:trailingSkip
- - ▪ dfdl:alignmentUnits

22.1.3 dfdl:sequence and dfdl:group (when reference is to a sequence)

- • *Parsing: hidden (xs:sequence only)*
- o dfdl:hiddenGroupRef
- • *Parsing: common*
- o dfdl:encoding

Comment [SMH560]: Remove byteOrder

DONE

- - ▪ 'UTF-16' 'UTF-16BE' 'UTF-16LE'
- - - ▪ dfdl:utf16Width
- o dfdl:ignoreCase
- • *Parsing: identification, framing & extraction*
- o dfdl:leadingSkip
- - ▪ dfdl:alignmentUnits
- o dfdl:alignment
- - ▪ *not "implicit"*
- - - ▪ dfdl:alignmentUnits
- o dfdl:initiator
- o dfdl:sequenceKind
- o dfdl:initiatedContent
- o dfdl:separator
- - ▪ dfdl:separatorPosition
- - ▪ dfdl:separatorSuppressionPolicy

Comment [SMH561]: Errata 3.14 changes
the name
DONE

- o dfdl:terminator
- - ▪ dfdl:documentFinalTerminatorCanBeMissing

- o dfdl:trailingSkip
    - ▪ dfdl:alignmentUnits

22.1.4 **dfdl:choice and dfdl:group (when reference is to a choice)**

- • *Parsing: common*
    - o dfdl:encoding
        - ▪ 'UTF-16' 'UTF-16BE' 'UTF-16LE'
            - ▪ dfdl:utf16Width
    - o dfdl:ignoreCase
- • *Parsing: identification, framing & extraction*
    - o dfdl:leadingSkip
        - ▪ dfdl:alignmentUnits
    - o dfdl:alignment
        - ▪ *not "implicit"*
            - ▪ dfdl:alignmentUnits
    - o dfdl:initiator
    - o dfdl:choiceLengthKind
        - ▪ *"explicit"*
            - ▪ dfdl:choiceLength
    - o dfdl:initiatedContent
    - o dfdl:choiceBranchRef
    - o dfdl:elementID
    - o dfdl:terminator
        - ▪ dfdl:documentFinalTerminatorCanBeMissing
    - o dfdl:trailingSkip
        - ▪ dfdl:alignmentUnits

22.2 **Unparsing**

The following list gives the order in which DFDL properties are examined when the DFDL unparser is positioned at a particular component in the DFDL Infoset, and about to unparsed and thereby create the bitstream which is the representation of that component.

22.2.1 **dfdl:element (simple) and dfdl:simpleType**

- • *Unparsing: calculated value (does not apply to dfdl:simpleType or to global elements)*
    - o dfdl:inputValueCalc (if set then element is ignored)
    - o dfdl:outputValueCalc
- • *Unparsing: common*
    - o dfdl:outputNewLine
    - o dfdl:encoding
        - ▪ 'UTF-16' 'UTF-16BE' 'UTF-16LE'

Comment [SMH562]: Missing the new choiceBranchRef stuff

Added below. Plz review
OPEN

Comment [SMH563]: Remove byteOrder
DONE

Comment [m564]: OPEN

Review plz

SMH: dfdl:elementID belongs on dfdl:element, not here

MJB: but this is WHEN you have to examine it. If we put it on the precedence tree for element, the timing of when it is needed to be examined is lost. (part of the overall problem with this precedence section)

Comment [SMH565]: Errata 3.2 and 2.7
DONE

Comment [SMH566]: Now that byte order is only applicable to binary reps, it could be moved down

MOVED. DONE

- ▪ dfdl:utf16Width
    - o dfdl:fillByte
- • *Unparsing: repeats (does not apply to dfdl:simpleType or to global elements)*
    - o (maxOccurs > 1 or unbounded) or (minOccurs = 0 and maxOccurs = 1)
        - ▪ dfdl:occursCountKind
            - ▪ *"expression"*
                - ▪ dfdl:occursCount
            - ▪ *"fixed", "implicit"*
                - ▪ minOccurs
                - ▪ maxOccurs
            - ▪ *"parsed"*
            - ▪ *"stopValue"*
                - ▪ dfdl:occursStopValue

> **Comment [SMH567]:** Add implicit DONE

- • *Unparsing: conversion*
    - o dfdl:useNilForDefault *(does not apply to dfdl:simpleType)*
        - ▪ *"true"*
            - ▪ *None*
        - ▪ *"false"*
            - ▪ XSDL default *or* fixed
    - o XSDL nillable *(does not apply to dfdl:simpleType)*
        - ▪ dfdl:nilKind
            - ▪ *"literalValue", "logicalValue", "literalCharacter"*
                - ▪ dfdl:nilValue
    - o xs:type
        - ▪ *"Number"*
            - ▪ dfdl:decimalSigned
            - ▪ dfdl:representation
                - ▪ *"text"*
                    - ▪ dfdl:textNumberRep
                        - ▪ *"standard"*
                            - ▪ dfdl:textNumberPattern
                            - ▪ dfdl:textStandardBase
                            - ▪ dfdl:textStandardDecimalSeparator
                            - ▪ dfdl:textStandardGroupingSeparator
                            - ▪ dfdl:textStandardExponentRep

> **Comment [SMH568]:** Errata 2.46 changes name **DONE**

- dfdl:textNumberCheckPolicy
- dfdl:textStandardInfinityRep
- dfdl:textStandardNaNRep

  Comment [SMH569]: NaN
  DONE

- dfdl:textNumberRounding
  - "*explicit*"
    - dfdl:textNumberRoundin gMode
    - dfdl:textNumberRoundin gIncrement
- dfdl:textStandardZeroRep

- *"zoned"*
  - *d*fdl:textNumberPattern
  - dfdl:textNumberCheckPolicy
  - dfdl:textNumberRounding
    - "*explicit*"
      - dfdl:textNumberRoundin gMode
      - dfdl:textNumberRoundin gIncrement
  - dfdl:textZonedSignStyle
- dfdl:textBidi
  - dfdl:textBidiOrdering

    Comment [SMH570]: Errata 2.60 changes name **DONE**

  - dfdl:textBiDiOrientation
  - dfdl:textBidiNumeralShapes
- *"binary"*
  - dfdl:byteOrder

    Comment [SMH571]: Byte Order introduced here.
    DONE

  - *xs:decimal and restrictions*
    - dfdl:binaryNumberRep
      - *"packed"*
        - dfdl:binaryPackedSignC odes
        - dfdl:binaryDecimalVirtu alPoint
      - *"bcd", "ibm4690Packed"*
        - dfdl:binaryDecimalVirtu alPoint
      - *"binary"*
        - dfdl:binaryDecimalVirtu alPoint
  - *xs:float, xs:double*

- dfdl:binaryFloatRep
- *"String"*
    - dfdl:textBidi
        - dfdl:textBidiOrdering
        - dfdl:textBiDiOrientation
        - dfdl:textBidiSymmetric
        - dfdl:textBidiShaped
- *"Calendar"*
    - dfdl:representation
        - *"text"*
            - dfdl:calendarPatternKind "*explicit*"
                - dfdl:calendarPattern
            - dfdl:calendarCheckPolicy
            - dfdl:calendarTimeZone
            - dfdl:calendarObserveDST
            - dfdl:calendarFirstDayOfWeek
            - dfdl:calendarDaysInFirstWeek
            - dfdl:calendarLanguage
            - dfdl:textBidi
                - dfdl:textBidiOrdering
                - dfdl:textBiDiOrientation
                - dfdl:textBidiSymmetric
                - dfdl:textBidiShaped
        - *"binary"*
            - dfdl:byteOrder
            - dfdl:binaryCalendarRep
                - *"packed"*
                    - dfdl:packedDecimalSignCodes
                    - dfdl:decimalVirtualPoint
                    - dfdl:calendarPatternKind
                      - "*explicit*"
                          - dfdl:calendarPattern
                    - dfdl:calendarCheckPolicy
                    - dfdl:calendarTimeZone
                    - dfdl:calendarObserveDST
                    - dfdl:calendarFirstDayOfWeek
                    - dfdl:calendarDaysInFirstWeek

Comment [SMH572]: Errata 2.60 changes name **DONE**

Comment [SMH573]: Errata 2.60 changes name **DONE**

Comment [SMH574]: Errata 2.60 changes name **DONE**

Comment [SMH575]: Errata 2.60 changes name **DONE**

- - - dfdl:calendarCenturyStart
  - - *"bcd", "ibm4690Packed"*
    - - dfdl:decimalVirtualPoint
    - - dfdl:calendarPatternKind
      - "*explicit*"
        - - dfdl:calendarPattern
    - - dfdl:calendarCheckPolicy
    - - dfdl:calendarTimeZone
    - - dfdl:calendarObserveDST
    - - dfdl:calendarFirstDayOfWeek
    - - dfdl:calendarDaysInFirstWeek
    - - dfdl:calendarCenturyStart
  - - *"binarySeconds", "binaryMilliseconds"*
    - - dfdl:binaryCalendarEpoch
- - *"Opaque"*
- - *"Boolean"*
  - - dfdl:representation
    - - *"text"*
      - - dfdl:textBooleanTrueRep
      - - dfdl:textBooleanFalseRep
      - - dfdl:textBidi
        - - dfdl:textBidiOrdering
        - - dfdl:textBiDiOrientation
        - - dfdl:textBidiSymmetric
        - - dfdl:textBidiTextShaped
    - - *"binary"*
      - - dfdl:byteOrder
      - - dfdl:binaryBooleanTrueRep
      - - dfdl:binaryBooleanFalseRep

> **Comment [SMH576]:** Errata 2.60 changes name **DONE**

> **Comment [SMH577]:** Errata 2.60 changes name **DONE**

- *Unparsing: insertion & framing*
  - dfdl:leadingSkip
    - - dfdl:alignmentUnits
  - dfdl:alignment
    - *not "implicit"*
      - - dfdl:alignmentUnits
  - dfdl:representation *"text" or xs:simpleType 'string'*
    - - dfdl:escapeSchemeRef

> **Comment [m578]:** Errata 2.74 Done.

- dfdl:lengthKind
  - *"implicit"*
    - xs:maxLength *or* dfdl:textBooleanTrueRep/dfdl:textBooleanFalseRep
    - dfdl:lengthUnits
    - dfdl:textPadKind
      - dfdl:textStringPadCharacter, dfdl:textNumberPadCharacter, dfdl:textBooleanPadCharacter or dfdl:textCalendarPadCharacter
      - dfdl:textStringJustification, dfdl:textNumberJustification, dfdl:textBooleanJustification or dfdl:textCalendarJustification
    - dfdl:truncateSpecifiedLengthString
  - *"explicit"*
    - *not* expression
      - dfdl:length
      - dfdl:truncateSpecifiedLengthString
    - expression
      - xs:minLength *or* dfdl:textOutputMinLength
    - dfdl:lengthUnits
    - dfdl:textPadKind
      - dfdl:textStringPadCharacter, dfdl:textNumberPadCharacter, dfdl:textBooleanPadCharacter or dfdl:textCalendarPadCharacter
      - dfdl:textStringJustification, dfdl:textNumberJustification, dfdl:textBooleanJustification or dfdl:textCalendarJustification
  - *"prefixed"*
    - dfdl:prefixLengthType
    - dfdl:prefixIncludesPrefixLength
    - dfdl:lengthUnits
    - dfdl:textPadKind
      - dfdl:textStringPadCharacter, dfdl:textNumberPadCharacter, dfdl:textBooleanPadCharacter or dfdl:textCalendarPadCharacter
      - dfdl:textStringJustification, dfdl:textNumberJustification,

> **Comment [SMH579]:** 2.100 (OPEN)
>
> Please review. Added 'explicit' since we might pad there if textOutputMinLength or xs:minLength is larger than the actual length of the infoset item.
>
> SMH: Reworked to put everything under 'explicit'

- dfdl:textBooleanJustification or
  dfdl:textCalendarJustification
  - xs:minLength *or* dfdl:textOutputMinLength
  - *"pattern", "delimited", "endOfParent"*
    - dfdl:textPadKind
      - dfdl:textStringPadCharacter,
        dfdl:textNumberPadCharacter,
        dfdl:textBooleanPadCharacter or
        dfdl:textCalendarPadCharacter
      - dfdl:textStringJustification,
        dfdl:textNumberJustification,
        dfdl:textBooleanJustification or
        dfdl:textCalendarJustification
      - xs:minLength *or* dfdl:textOutputMinLength
- dfdl:representation *"binary" or xs:simpleType 'hexBinary'*
  - dfdl:lengthKind
    - *"implicit"*
      - xs:maxLength *or* xs:simpleType
      - dfdl:lengthUnits
    - *"explicit"*
      - dfdl:length
      - dfdl:lengthUnits
    - *"prefixed"*
      - dfdl:prefixLengthType
      - dfdl:prefixIncludesPrefixLength
      - dfdl:lengthUnits
    - *"delimited", "endOfParent"*
      - *None*
- dfdl:initiator
  - dfdl:nilValueDelimiterPolicy *(does not apply to dfdl:simpleType)*
  - dfdl:emptyValueDelimiterPolicy
- dfdl:terminator
  - dfdl:nilValueDelimiterPolicy *(does not apply to dfdl:simpleType)*
  - dfdl:emptyValueDelimiterPolicy
- dfdl:trailingSkip
  - dfdl:alignmentUnits

22.2.2 dfdl:element (complex)

- *Unparsing: common*
  - dfdl:outputNewLine

| Comment [SMH580]: Errata 2.4. **DONE** |
| Comment [SMH581]: Need to decide if bcd/packed allowed with pattern, if not remove

DONE - removed. |
| Comment [SMH582]: Removed byteOrder DONE |

- o dfdl:encoding
    - ▪ 'UTF-16' 'UTF-16BE' 'UTF-16LE'
        - ▪ dfdl:utf16Width
- o dfdl:fillByte
- *Unparsing: nillable*
    - o XSDL nillable *(does not apply to dfdl:simpleType)*
        - ▪ dfdl:nilKind
            - ▪ *"literalValue"*
                - ▪ dfdl:nilValue (must be "%ES;")
- *Unparsing: repeats (does not apply to global elements)*
    - o (maxOccurs > 1 or unbounded) or (minOccurs = 0 and maxOccurs = 1)
        - ▪ dfdl:occursCountKind
            - ▪ *"expression"*
                - ▪ dfdl:occursCount
            - ▪ *"fixed"*, *"implicit"*
                - ▪ minOccurs
                - ▪ maxOccurs
            - ▪ *"parsed"*
- *Unparsing: insertion & framing*
    - o dfdl:leadingSkip
        - ▪ dfdl:alignmentUnits
    - o dfdl:alignment
        - ▪ *not "implicit"*
            - ▪ dfdl:alignmentUnits
    - o dfdl:initiator
        - ▪ dfdl:nilValueDelimiterPolicy
        - ▪ dfdl:emptyValueDelimiterPolicy
    - o dfdl:lengthKind
        - ▪ *"explicit"*
            - ▪ dfdl:length
            - ▪ dfdl:lengthUnits
        - ▪ *"prefixed"*
            - ▪ dfdl:prefixLengthType
            - ▪ dfdl:prefixIncludesPrefixLength
            - ▪ dfdl:lengthUnits
        - ▪ *"implicit", "pattern", "delimited", "endOfParent"*
            - ▪ *None*

Comment [SMH583]: Add implicit DONE	
Comment [SMH584]: Errata 3.10 adds nillable complex elements DONE - inserted before repeats.	
Comment [SMH585]: Errata 2.5. **DONE**	

- o dfdl:terminator
    - ▪ dfdl:nilValueDelimiterPolicy
    - ▪ dfdl:emptyValueDelimiterPolicy
- o dfdl:trailingSkip
    - ▪ dfdl:alignmentUnits

22.2.3 dfdl:sequence and dfdl:group (when reference is a sequence)

- • *Unparsing: hidden (xs:sequence only)*
    - o dfdl:hiddenGroupRef
- • *Unparsing: common*
    - o dfdl:outputNewLine
    - o dfdl:encoding
        - ▪ 'UTF-16' 'UTF-16BE' 'UTF-16LE'
            - ▪ dfdl:utf16Width
    - o dfdl:fillByte
- • *Unparsing: insertion & framing*
    - o dfdl:leadingSkip
        - ▪ dfdl:alignmentUnits
    - o dfdl:alignment
        - ▪ *not "implicit"*
            - ▪ dfdl:alignmentUnits
    - o dfdl:initiator
    - o dfdl:separator
        - ▪ dfdl:separatorPosition
        - ▪ dfdl:separatorSuppressionPolicy
    - o dfdl:terminator
    - o dfdl:trailingSkip
        - ▪ dfdl:alignmentUnits

22.2.4 dfdl:choice and dfdl:group (when reference is a choice)

- • *Unparsing: common*
    - o dfdl:outputNewLine
    - o dfdl:encoding
        - ▪ 'UTF-16' 'UTF-16BE' 'UTF-16LE'
            - ▪ dfdl:utf16Width
    - o dfdl:fillByte
- • *Unparsing: insertion & framing*
    - o dfdl:leadingSkip

Comment [SMH586]: Removed ByteOrder. DONE

Comment [SMH587]: Errata 3.10 adds nillable complex elements

Only elements can be nillable, so it was done there. Can't have nillable sequences and choices, only elements.

DONE

Comment [SMH588]: Errata 3.14 changes the name
DONE

Comment [SMH589]: Removed ByteOrder
DONE

- dfdl:alignmentUnits
  - o dfdl:alignment
    - *not "implicit"*
      - dfdl:alignmentUnits
  - o dfdl:initiator
  - o dfdl:choiceLengthKind
    - *explicit"*
      - dfdl:choiceLength
  - o dfdl:terminator
  - o dfdl:trailingSkip
    - dfdl:alignmentUnits

23. Expression language

The DFDL expression language allows the processing of values conforming to the data model defined in the DFDL Infoset. It allows properties in the DFDL schema to be dependent on the value of an occurrence of an element or the value of a DFDL variable. For example the length of the content of an element can be made dependent on the value of another element in the document.

The main uses of the expression language are as follows:

1. When a DFDL property needs to be set dynamically at parse time from the value of one or more elements of the data. Properties such as initiator, terminator, length, occursCount and separator accept an expression.

2. In a dfdl:assert annotation

3. In a dfdl:discriminator annotation to resolve uncertainty when parsing

4. In a dfdl:inputValueCalc property to derive the value of an element in the logical model that doesn't exist in the physical data.

5. In a dfdl:outputValueCalc property to compute the value of an element on unparsing.

6. As the value in a dfdl:setVariable annotation or the dfdl:defaultValue in a dfdl:defineVariable or dfdl:newVariableInstance.

The DFDL expression language is a subset of XPath 2.0 [XPath2]. DFDL uses a subset of XML schema and has a simpler information model, so only a subset of XPath 2.0 expressions is meaningful in DFDL Schemas. For example there are no attributes in DFDL so the attribute axis is not needed.

In addition, DFDL expressions never return node-sequences having more than one node. DFDL expressions either return a simple value, a node sequence containing exactly one node/value, or an empty node sequence.

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. There are two exceptions to this: the dfdl:occursCount and dfdl:occursCountWithDefault functions take a path ending in the name of an array or optional element.

DFDL implementations MUST comply with the error code behaviour in Appendix G of the XPath 2.0 spec 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

Implementation Note: 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:

- Wrap path locations in a call to fn:exactly-one() except when the path location occurs within certain functions which operate on arrays
- Check for the disallowed use of those XPath 2.0 functions that are not in the DFDL subset

23.1   Expression Language Data Model

The DFDL expression language operates on the DFDL infoset with the addition of the hidden elements. That is, it operates on the *augmented* infoset.

Comment [m590]: It's nilValue singular, and it no longer accepts an expression
DONE

Comment [SMH591]: No dfdl: as it's just an attribute

Comment [m592]: Errata 2.102. *Section 23*. State that it is a schema definition error if an array element appears…
DONE

Comment [m593]: **Errata 2.114**. *Section 23*. DFDL implementations MUST comply with the error code behaviour …
DONE

Comment [m594]: **Errata 2.113**. *Section 23*. DFDL implementations may use off-the-shelf …
DONE

Comment [m595]: Add this function (OPEN)

During parsing, expressions can reference any element preceding the current position. During parsing only a limited degree of reference to elements following the current position is allowed. Forward reference can occur only in the context of a dfdl:discriminator annotation and implementations may have varying ability to support forward reference.

During unparsing, expressions can reference any element either preceding or following the current element in the data stream.

Implementations may have specific limitations on the use of these forward or backward reference, or may provide controls for bounding the reach of such references. These mechanisms are beyond the scope of this specification.

23.2 Variables

A variable is a binding between a (qualified) name and a (typed) value. Variables are defined using the dfdl:defineVariable annotation (see 7.7); defining a variable causes an initial instance also to be created. Further instances of variables are created using the dfdl:newVariableInstance annotation. Instances of variables are assigned a value using the dfdl:setVariable annotation. Variables are referenced in expressions by preceding the QName with '$'.

This section describes the semantics of variables. Any implementation consistent with the behavior described here is acceptable.

The memory where the information about a variable is stored during DFDL processing is called the *variable memory*. A variable is a name that is associated with a storage tuple in the variable memory.

Specifically, the variable memory contains:

- a counter used to generate locations for new tuples. Initial value is 1.
- an ordered list of locations. Each location contains a tuple of values:
    - o has-been-set flag. This Boolean is originally false. dfdl:setVariable changes this flag to true.
    - o has-been-referenced flag. This Boolean is originally false. Evaluation of an expression that uses the variable value changes the value to true.
    - o has-value flag. This Boolean is originally true if the dfdl:defineVariable or dfdl:newVariableInstance annotation has a default value specified, or if a default value has been supplied externally. Otherwise it is false, but is set to true if a dfdl:setVariable annotation is processed.
    - o typeID. This string is a type identifier taken from the type specified in the dfdl:defineVariable annotation.
    - o value. This is a typed value, or the distinguished value "unknown". The type of the value must correspond to the typeID. The value is optionally specified in dfdl:defineVariable or dfdl:newVariableInstance annotations in which case we refer to it as the *default value* for the variable. A default value may also be provided by the DFDL processor when the variable is defined with external="true".

The variable memory is initialized when a dfdl:defineVariable annotation is encountered.

Each time a dfdl:newVariableInstance annotation is encountered, the parser captures the current value of the counter from the variable memory. It then creates a new variable memory where the location counter's value is one greater, and where the list of locations has been augmented with a new tuple at the location given by the prior value of the location counter. The tuple is initialized based on the specifics of the dfdl:defineVariable annotation.

23.2.1 Rewinding of Variable Memory State

Upon exit of the scope where the new variable instance was created, the newly created variable memory is discarded and the prior variable memory is restored.

Note that the above algorithm insures that each time a dfdl:newVariableInstance is encountered, a fresh location is initialized for it, and once the scope containing that variable goes out of scope, the instance tuple for the variable can no longer be reached. A different variable instance tuple may now be visible if there is one still in an enclosing scope.

23.2.2 Variable Memory State Transitions

The flags in the variable memory tuples are interpreted and modified as follows:

DFDL annotation	before annotation processed			after annotation processed		
	has-been-set	has-been-referenced	has-value	has-been-set	has-been-referenced	has-value
defineVariable (without default or external value)	tuple doesn't exist			false	false	false
defineVariable (with default value)	tuple doesn't exist			false	false	true
defineVariable (with external value)	tuple doesn't exist			false	false	true
newVariableInstance (without default value)	tuple doesn't exist			false	false	false
newVariableInstance (with default value)	tuple doesn't exist			false	false	true
setVariable	tuple doesn't exist			schema definition error		
	false	false	false	true	false	true
	false	false	true	true	false	true (also value changed to new value)
	false	true	true	schema definition error – set after reference not allowed.		
	true	any	true	schema definition error - double set not allowed.		
reference variable (from DFDL expression)	tuple doesn't exist			schema definition error		
	false	false	false	schema definition error – undefined variable		
	any	any	true	false	true (value is returned)	true

Table 24 Variable memory states.

The above table describes a set of rules which might be abbreviated as:

- write once, read many
- no write after the value has been read

An exception to this behavior occurs whenever the DFDL processor backtracks because it is processing multiple arms of a choice or as a result of speculative parsing. In this case the variable state is also rewound.

It is a schema definition error if a dfdl:setVariable or a variable reference occurs and there is no corresponding variable name defined by a dfdl:defineVariable annotation.

It is a schema definition error if a dfdl:setVariable provides a value of incorrect type which does not correspond to the type specified by the dfdl:defineVariable.

It is a schema definition error if a variable reference in an expression is able to return a value of incorrect type for the evaluation of that expression. That is, DFDL - including the expressions contained in it - is a statically type-checkable language. DFDL implementations may issue these schema definition errors prior to processing time.

Even if the errors are detected at processing time, the errors associated with write-after-read, and double-write are schema definition errors because they indicate the schema is not properly designed to use variables consistent with their single-assignment behavior.

23.3   General Syntax

DFDL expressions follow the XPath 2.0 syntax rules but are always enclosed in curly braces "{" and "}".

When a property accepts either a DFDL string literal or a DFDL expression, and the value is a string literal starting with a "{" character, then "{{"must be used to escape the "{" character.

The syntax "{}" is a schema definition error as it results in an empty XPath 2.0 expression which is not legal. It is not the equivalent of setting the property to empty string.

Examples

```
{ /book/title }
{ $x+2 }
{ if (fn:exists(../field1))   then 1 else 0   }
```

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.

Additionally:

- o Every property that accepts an expression states exactly what the expression is expected to return. To ensure the returned value is of the correct type, an expression must use XPath constructors or the correct literal values.
- o What appears lexically as the syntax of an expression follows XPath 2.0 rules. Note specifically that this is not the same as XSDL default and fixed property lexical syntax. Specifically, XSDL default and fixed properties do not accept expressions. They are always interpreted as XML Schema string literals. See [XSDLV1] for details.
- o 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.
- o 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.
- o If expecting an expression to return a DFDL string literal, the returned value is always treated as a DFDL string literal.
- o Within an expression, a string is never interpreted as a DFDL string literal.

Comment [SMH598]: Errata 2.49 clarifies this. **DONE**

Comment [SMH599]: Errata 2.37: **DONE**

Comment [SMH600]: Errata 2.69 clarifies this and makes it consistent

DONE

Comment [m601]: Huh? What does "returned lexically" mean.

Changed to What appears lexically as an expression
DONE

Comment [m602]: Errata 2.83 as revised DONE

Comment [m603]: Errata 2.83 Done

23.4   DFDL Expression Syntax

Refer to XML Path Language (XPath) 2.0 [XPath2 for a description of XPath expressions

DFDL Expression	::=	"{" Expr "}"
Expr	::=	ExprSingle
ExprSingle	::=	IfExpr | OrExpr
IfExpr	::=	"if" "(" Expr ")" "then" ExprSingle "else" ExprSingle
OrExpr	::=	AndExpr ("or" AndExpr)*
AndExpr	::=	ComparisonExpr ("and" ComparisonExpr)*
ComparisonExpr	::=	AdditiveExpr ((ValueComp) AdditiveExpr)?
AdditiveExpr	::=	MultiplicativeExpr (("+" | "-") MultiplicativeExpr)*
MultiplicativeExpr	::=	UnaryExpr (("*" | "div" | "idiv" | "mod") UnaryExpr)*
UnaryExpr	::=	("-" | "+")* ValueExpr
ValueExpr	::=	PathExpr
ValueComp	::=	"eq" | "ne" | "lt" | "le" | "gt" | "ge"
PathExpr	::=	("/" RelativePathExpr?) | RelativePathExpr | FilterExpr
RelativePathExpr	::=	StepExpr (("/") StepExpr)*
StepExpr	::=	AxisStep
AxisStep	::=	(ReverseStep | ForwardStep) Predicate?
ForwardStep	::=	(ForwardAxis NodeTest) | AbbrevForwardStep
ForwardAxis	::=	("child" "::") | ("self" "::")
AbbrevForwardStep	::=	NodeTest | ContextItemExpr
ReverseStep	::=	(ReverseAxis NodeTest) | AbbrevReverseStep
ReverseAxis	::=	("parent" "::") ")

Comment [m604]: *Errata 2.101 Section 23.4.* The BNF 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.

DONE - went back to original grammar, then added changes per email thread Feb 11.2013

The changes have comment bubble notes on them There should be change bars, but for tables these are sometimes misleading.

Comment [m605]: Note change here.

Comment [m606]: Note change here.

Comment [m607]: Note change here (just the ? was added)

Comment [m608]: Note change here.

```
AbbrevReverseStep     ::=   ".."

NodeTest              ::=   NameTest

NameTest              ::=   QName

FilterExpr            ::=   PrimaryExpr Predicate?

Predicate             ::=   "[" Expr "]"

PrimaryExpr           ::=   Literal | VarRef |
                            ParenthesizedExpr | ContextItemExpr
                            | FunctionCall

Literal               ::=   NumericLiteral | StringLiteral

NumericLiteral        ::=   IntegerLiteral | DecimalLiteral |
                            DoubleLiteral

VarRef                ::=   "$" VarName

VarName               ::=   QName

ParenthesizedExpr     ::=   "(" Expr ")"

ContextItemExpr       ::=   "."

FunctionCall          ::=   QName "(" (ExprSingle (","
                            ExprSingle)*)? ")"
```

Comment [m609]: Note change here (just the ? was added)

Comment [m610]: Note change here (? was removed)

Table 25 DFDL Expression Language

Notes:

1. Only *If* and *path* expression types are supported

2. Only the *child*, *parent*, and *self* axes are supported

3. Predicates are only used to index arrays and so must be integer expressions otherwise a schema definition error occurs

4. A subset of the XPath 2.0 operators are supported

23.5 Constructors, Functions and Operators

In the function signatures below a '?' following an argument name, argument type or result type indicates that the argument/result can be a node or value of the expected type or it can have no value.

Comment [m611]: DONE

Need a statement about errors.

We say this language is statically type-checkable…

So type errors are SDE, conversion errors are PE.

23.5.1 Constructor Functions for XML Schema Built-in Types

The arguments to the constructors are all of type xs:anyAtomicType. Since the expression language can be statically type checked, it is a schema definition error if the type of the argument is not one of the DFDL-supported subtypes of xs:anyAtomicType.

Comment [m612]: Added phrase about SDE and processing error.

Comment [SMH613]: Subtypes of

However, many statically type-correct values will still not be convertible to the result type.It is a processing error if the supplied argument value is not convertible to the constructed type.

The following constructor functions for the built-in types are supported:

- xs:string($arg as xs:anyAtomicType) as xs:string

- xs:boolean($arg as xs:anyAtomicType) as xs:boolean

- xs:decimal($arg as xs:anyAtomicType) as xs:decimal

- xs:float($arg as xs:anyAtomicType) as xs:float

- xs:double($arg as xs:anyAtomicType) as xs:double

- xs:dateTime($arg as xs:anyAtomicType) as xs:dateTime

- xs:time($arg as xs:anyAtomicType) as xs:time

- xs:date($arg as xs:anyAtomicType) as xs:date

- xs:hexBinary($arg as xs:anyAtomicType) as xs:hexBinary

- xs:integer($arg as xs:anyAtomicType) as xs:integer

- xs:long($arg as xs:anyAtomicType) as xs:long

- xs:int($arg as xs:anyAtomicType) as xs:int

- xs:short($arg as xs:anyAtomicType) as xs:short

- xs:byte($arg as xs:anyAtomicType) as xs:byte

- xs:nonNegativeInteger($arg as xs:anyAtomicType) as xs:nonNegativeInteger

- xs:unsignedLong($arg as xs:anyAtomicType) as xs:unsignedLong

- xs:unsignedInt($arg as xs:anyAtomicType) as xs:unsignedInt

- xs:unsignedShort($arg as xs:anyAtomicType) as xs:unsignedShort

- xs:unsignedByte($arg as xs:anyAtomicType) as xs:unsignedByte

> **Comment [SMH614]:** Any reason why these are in fixed width font, compared to the later tables?

A special constructor function is provided for constructing a `xs:dateTime` value from an `xs:date` value and an `xs:time` value.

- fn:dateTime($arg1 as xs:date, $arg2 as xs:time) as xs:dateTime

23.5.2 Standard XPath Functions

> **Comment [SMH615]:** All functions should state arguments

23.5.2.1 Boolean functions

The following additional constructor functions are defined on the boolean type.

Function	Meaning
fn:true()	Constructs the xs:boolean value 'true'.
fn:false()	Constructs the xs:boolean value 'false'.

Table 26 Boolean functions

The following functions are defined on boolean values:

Function	Meaning
fn:not($arg?)	If `$arg` is the empty sequence or nil, `fn:not` returns `true`.
	If `$arg` is a sequence containing a node, `fn:not` returns `false`.
	If `$arg` is a value of type `xs:boolean` or a derived from `xs:boolean`, `fn:not` returns the boolean inverse of `$arg`.
	If `$arg` is a value of type `xs:string` or a type derived from `xs:string`, `fn:not` returns `true` if the operand value has zero length; otherwise it returns `false`.
	If `$arg` is a value of any numeric type or a type derived from a numeric type, `fn:not` returns `true` if the operand value is `NaN` or is numerically equal to zero; otherwise it returns `false`.
	In all other cases, `fn:not` raises a processing error.
	Inverts the xs:boolean value of the argument.

Comment [SMH616]: Can only return one item else schema definition error.

Comment [m617]: Fixed Wording DONE

Comment [m618]: We don't have the fn:boolean xpath function, so the definition of it was inlined here, with inverted results to implement the NOT semantics.

Comment [SMH619]: Mixture of fonts in this table but not following tables.

Table 27 Boolean functions

23.5.2.2 Numeric Functions

The following functions are defined on numeric types. Each function returns a value of the same type as the type of its argument. The argument must be convertible to a number type.

Function	Meaning
fn:abs($arg as numeric)	Returns the absolute value of the argument.
fn:ceiling($arg as numeric)	Returns the smallest number with no fractional part that is greater than or equal to the argument.
fn:floor($arg as numeric)	Returns the largest number with no fractional part that is less than or equal to the argument.
fn:round($arg as numeric)	Rounds to the nearest number with no fractional part.
fn:round-half-to-even($arg as numeric) fn:round-half-to-even($arg as numeric, $precision as xs:integer)	Takes a number and a precision and returns a number rounded to the given precision. If the fractional part is exactly half, the result is the number whose least significant digit is even.

Comment [SMH620]: In fn:

Comment [SMH621]: What does the '?' signify after the type of an argument.

MJB: From the xpath spec:

Some functions accept a single value or the empty sequence as an argument and some may return a single value or the empty sequence. This is indicated in the function signature by following the parameter or return type name with a question mark: "?", indicating that either a single value or the empty sequence must appear.

This means many/most of our functions should NOT have the ?. Because we don't want everything to be legal and return empty node sequences when the arguments are incorrect (which is what XPath 2.0 seems to do.)

Expressions, including arguments, behave as if fn:exactly-one(…) was wrapped around them because empty node sequences are not accepted.

Table 28 Numeric Functions

23.5.2.3 String Functions

The following functions are defined on values of type `xs:string` and types derived from it.

Function	Meaning

Function	Meaning
fn:concat($arg1 as xs:anyAtomicType, $arg2 as xs:anyAtomicType, ...)	Concatenates two or more xs:anyAtomicType arguments cast to xs:string.
fn:substring($sourceString as xs:string, $startingLoc as xs:double) fn:substring($sourceString as xs:string, $startingLoc as xs:double, $length as xs:double)	Returns the xs:string located at a specified place within an argument xs:string.
fn:string-length($arg as xs:string)	Returns the length of the argument as an xs:integer
fn:upper-case($arg as xs:string)	Returns the upper-cased value of the argument.
fn:lower-case($arg as xs:string)	Returns the lower-cased value of the argument.

Function	Meaning
fn:contains($arg1 as xs:string, $arg2 as xs:string) fn:contains($arg1 as xs:string, $arg2 as xs:string, $collation as xs:string)	Returns xs:boolean indicating whether one xs:string contains another xs:string. A collation may be specified.
fn:starts-with($arg1 as xs:string, $arg2 as xs:string) fn:starts-with($arg1 as xs:string, $arg2 as xs:string, $collation as xs:string)	Returns xs:boolean indicating whether the value of one xs:string begins with the collation units of another xs:string. A collation may be specified.
fn:ends-with($arg1 as xs:string, $arg2 as xs:string) fn:ends-with($arg1 as xs:string, $arg2 as xs:string, $collation as xs:string)	Returns xs:boolean indicating whether the value of one xs:string ends with the collation units of another xs:string. A collation may be specified.
fn:substring-before($arg1 as xs:string, $arg2 as xs:string) fn:substring-before($arg1 as xs:string, $arg2 as xs:string, $collation as xs:string)	Returns the collation units of one xs:string that precede in that xs:string the collation units of another xs:string. A collation may be specified.
fn:substring-after($arg1 as xs:string, $arg2 as xs:string) fn:substring-after($arg1 as xs:string, $arg2 as xs:string, $collation as xs:string)	Returns the collation units of xs:string that follow in that xs:string the collation units of another xs:string. A collation may be specified.

Table 29 String Functions

23.5.2.4 Date, Time functions

Function	Meaning
fn:year-from-dateTime($arg as xs:dateTime)	Returns the year from an xs:dateTime value.
fn:month-from-dateTime($arg as xs:dateTime)	Returns the month from an xs:dateTime value.
fn:day-from-dateTime($arg as xs:dateTime)	Returns the day from an xs:dateTime value.
fn:hours-from-dateTime($arg as xs:dateTime)	Returns the hours from an xs:dateTime value.
fn:minutes-from-dateTime($arg as xs:dateTime)	Returns the minutes from an xs:dateTime value.
fn:seconds-from-dateTime($arg as xs:dateTime)	Returns the seconds from an xs:dateTime value.
fn:year-from-date($arg as xs:date)	Returns the year from an xs:date value.
fn:month-from-date($arg as xs:date)	Returns the month from an xs:date value.
fn:day-from-date($arg as xs:date)	Returns the day from an xs:date value.
fn:hours-from-time($arg as xs:time)	Returns the hours from an xs:time value.
fn:minutes-from-time($arg as xs:time)	Returns the minutes from an xs:time value.
fn:seconds-from-time($arg as xs:time)	Returns the seconds from an xs:time value.

Table 30 Date, Time functions

23.5.2.5 Node Sequence Test Functions

The following functions are defined on sequences. (Note that DFDL v1.0 does not support sequences of length > 1.)

Function	Meaning
fn:empty($arg?)	Indicates whether or not the provided sequence is empty.
fn:exists($arg?)	Indicates whether or not the provided sequence is not empty.
fn:exactly-one($arg?)	True if the provided sequence contains exactly one node/value.

Table 31 Sequences functions

> **Comment [SMH622]:** Should we clarify what 'sequence' means here?

23.5.2.6 **Node functions**

This section discusses functions and operators on nodes.

Function	Meaning
fn:name() fn:name($arg?)	Returns the name of the context node (".") or the specified node as an xs:string. Returns zero-length string if the arg is an empty sequence. Returns an xs:string in the form of an xs:QName otherwise.
fn:local-name() fn:local-name($arg)	Returns the local name of the context node or the specified node as an xs:NCName.
fn:namespace-uri() fn:namespace-uri($arg)	Returns the namespace URI as an xs:string for the argument node or the context node if the argument is omitted. Returns empty string if the argument/context node is in no namespace.

Table 32 Node functions

23.5.3 **DFDL Functions**

Function	Meaning
dfdl:contentLength($node, $lengthUnits)	Returns the length of the supplied node's SimpleContent region for elements of simple type, or ComplexContent region for elements of complex type. These regions are defined in Section 9.2 DFDL Data Syntax Grammar. The value is returned as an xs:unsignedLong. The second argument is 'bytes', 'characters', or 'bits' and determines the units of length.
dfdl:valueLength($node, $lengthUnits)	Returns the length of the supplied node's ***SimpleValue or NilLogicalValue*** region for elements of simple type, or ComplexContent region for elements of complex type. These regions are defined in Section 9.2 DFDL Data Syntax Grammar. The value is returned as an xs:unsignedLong. For simple types, the valueLength() function returns a length which excludes any padding or filling. The second argument is 'bytes', 'characters', or 'bits' and determines the units of length.
dfdl:testBit($data, $bitPos)	Returns Boolean true if the bit number given by $bitPos is set on in the byte given by $data, otherwise

Comment [SMH623]: Does this return xs:string or xs:QName ? It can't do both.

Comment [m624]: Added xs:string in the form of…
DONE

Comment [m625]: (OPEN)

Improve. Can the context be something other than a node? or is that an SDE (static type error)

Comment [SMH626]: Need arguments defined, as these are not XPath 2.0.
DONE

Comment [m627]: Errata 2.96. DONE

Comment [m628]: Errata 2.96 Done

Comment [m629]: Errata 3.18 Done

Comment [m630]: 3.18. (OPEN)

Please review: Updated this to refer to the grammar regions.

Comment [m633]: 3.18 (OPEN)

Please review: is this right for complex types there is no difference between contentLength and valueLength ??

Comment [m631]: Errata 3.18 Done

Comment [m632]:

Comment [m634]: DONE: I think this is not well defined. How long this is depends on representation properties doesn't it? E.g., 1000 might be 5 characters if there is a grouping separator specified.

Resolved: change language to specifically refer to the grammar regions to make this unambiguous. (Yes 1,000 is more than a pure 'value', it's the SimpleRepresentation region.)

Comment [m635]: Errata 2.84 dfdl:property was removed. DONE.

Comment [m636]:

Comment [m637]: (DONE – was abuot dfdl:property which was removed.)

Does this evaluate the property if it is an expression?

(Suggest YES)

Are entities substituted if this expression returns a string literal value (e.g., an expression for terminator property?)

(Suggest NO)

Function	Meaning	
	returns Boolean false.	
dfdl:setBits($bit1, $bit2, ... $bit8)	Returns an unsigned byte being the value of the bit positions provided by the Boolean arguments, where true=1, false=0. The number of arguments must be 8.	
dfdl:occursCountWithDefault($arrayPath)	Returns the count of the number of occurrences including the effect of defaulting as an xs:long. The $arrayPath argument must end in the name of an element without any index/predicate.	**Comment [m638]:** Note: changed to xs:long. DONE **Comment [m639]:** Added clarification. DONE
dfdl:occursCount($arrayPath)	Returns the count of the number of occurrences excluding the effect of defaulting as an xs:long. The $arrayPath argument must end in the name of an element without any index/predicate.	**Comment [m640]:** Note: changed to xs:long. DONE **Comment [m641]:** Added clarification. DONE
dfdl:occursIndex()	Returns the position of the current item within an array as an xs:long. The first element is at position 1.	**Comment [SMH642]:** Indexed from 0 or 1 ? Base 1. DONE **Comment [m643]:** Changed to xs:long DONE
dfdl:checkConstraints($node)	Returns boolean true if the specified node value satisfies the XML schema facet constraints that are associated with it. Returns false if the specified node does not meet the constraints or does not exist. The facets that are checked are • minLength, maxLength • pattern • enumeration • maxInclusive, maxExclusive, minExclusive, minInclusive • totalDigits • fractionDigits See Section 5.2 for which facets are checked for each simple type. Additionally the fixed attribute is checked.	**Comment [m644]:** 2.111. Done.
dfdl:stringLiteralFromString($arg)	Returns a DFDL string literal constructed from the $arg string argument. If $arg contains any '%' and/or space characters, then the	

Function	Meaning
	return value replaces each '%' with '%%' and each space with '%SP;', otherwise $arg is returned unchanged.
	Use this function when the value of a DFDL property is obtained from the data stream using an expression, and the type of the property is DFDL String Literal or List of DFDL String Literals, and the values extracted from the data stream could contain '%' or space characters. If the data already contains DFDL entities, this function should not be used.
dfdl:containsEntity($arg)	Returns a Boolean indicating whether the $arg string argument contains one or more DFDL entities.

Comment [m645]: Errata 2.85. Done

Table 33 DFDL Functions

Notes:

dfdl:valueLength(path, lengthUnits) - returns the value length which excludes any padding or filling which might be added for a *specified length*

Comment [m646]: (OPEN) Is this well defined even?

It seems an abstract value other than a string doesn't have an inherent length. E.g., the number 1000 is an abstract concept. It its content takes up a minimum of 4 characters base 10, but 3 in base 16. Depending on textNumberPattern, it may need a grouping separator, which increases the size of the content region in our grammar.

Do we really want paddedContentLength and unpaddedContentLength ?

If the element declaration in the DFDL schema corresponding to the infoset item is not potentially represented, then the unpadded length is defined to be 0.

The value length includes the length contributions from introduced escape characters needed to escape contained delimiters (if such are defined, and will appear in the output representation).

The value length is also a function of the dfdl:encoding property. Multi-byte and variable-width character set encodings will commonly contribute more bytes to the value length than a single-byte character set would.

The value length is computed from the DFDL infoset value, ignoring the dfdl:length or dfdl:textOutputMinLength property. Other DFDL properties which affect the length of a text or binary representation are respected, it is only an explicit length which is ignored.

For a complex type, this means a bottom up totaling of the dfdl:contentLength() of all the contents and framing of the complex type.

dfdl:contentLength(path, lengthUnits) – returns the length of the content of the infoset data item as identified by the path argument. This includes padding or filling or truncation which might be carried out for a *specified length* item.

Comment [SMH647]: Typo

If the element declaration in the DFDL schema corresponding to the infoset item is not potentially represented (e.g., has an dfdl:inputValueCalc property), then the length is defined to be 0.

When unparsing with dfdl:lengthKind="explicit", the calculation of dfdl:contentLength() returns the value of the dfdl:length property.

For both dfdl:contentLength() and dfdl:valueLength(), the content length excludes any alignment filling as well as excluding any leading or trailing skip bytes. That is, the returned length is about the length of the content, and not about the position of that content in the output data stream.

24. DFDL Regular Expressions

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.

A DFDL regular expression may be either a PERL regular expression [PERLRE] or a Java® regular expression [JAVARE]. For portability it is recommended that use is restricted to a common subset of the PERL and Java syntax.

DFDL regular expressions do not interpret DFDL entities.

Comment [m648]: Errata 2.86 Done

25. Security Considerations

All locations must be properly initialized before writing so as to prevent accidental (or purposeful) transmission of data in the unused parts of data formats. Even when a DFDL description does not specify that data should be written to a particular part of the output representation, a defined pattern should always be written.

When unparsing data it is a schema definition error if the representation properties that control filling and padding are not defined by the DFDL schema. The DFDL processor must fail if they are not defined so that it is certain no region of the output data has unspecified contents.

If regions within a DFDL-described data object are encrypted, then when decrypting them proper means must be used to assure secure passage of passwords to the decrypting software. Such means are beyond the scope of the DFDL language specification.

In addition, if encryption passwords/keys are stored in DFDL schema-described data, then proper means must be used to assure that the decrypted form of these passwords is not revealed. Such means are beyond the scope of the DFDL language specification.

26. Authors and Contributors

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

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

Alan W. Powell,
apowell888@googlemail.com

We greatly acknowledge the contributions made to this document by the following and all the other people who provided constructive and valuable input in the group discussions.

Tim Kimber, IBM Software Group, Hursley, UK

Suman Kalia, IBM Software Group, Markham, Ontario, Canada

Stephanie Fetzer, IBM Software Group, Charlotte, USA

Martin Westhead, Avaya, Milpitas, CA, USA

James Myers, NCSA, Urbana-Champaign, IL, USA

Tom Sugden, EPCC

Tara Talbot, PNNL, Richland, WA, USA

Robert McGrath, NCSA, Urbana-Champaign, IL, USA

Geoff Judd, IBM Software Group, Hursley, UK

Dewey M. Sasser, MA, USA

David A. Loose, IBM Software Group, Westborough, MA, USA

Eric S. Smith, IBM Software Group, Westborough, MA, USA

Kristoffer H. Rose, IBM Research, Hawthorne, NY, USA

Simon Parker, Polar Lake, UK

Peter A. Lambros, IBM Software Group, Hursley, UK

Dave Glick, USA

Steve Marting, Progeny, USA

Alejandro Rodriguez, NCSA, Urbana-Champaign, IL, USA

27. Intellectual Property Statement

The OGF takes no position regarding the validity or scope of any intellectual property or other rights that might be claimed to pertain to the implementation or use of the technology described in this document or the extent to which any license under such rights might or might not be available; neither does it represent that it has made any effort to identify any such rights. Copies of claims of rights made available for publication and any assurances of licenses to be made available, or the result of an attempt made to obtain a general license or permission for the use of such proprietary rights by implementers or users of this specification can be obtained from the OGF Secretariat.

The OGF invites any interested party to bring to its attention any copyrights, patents or patent applications, or other proprietary rights which may cover technology that may be required to practice this recommendation. Please address the information to the OGF Executive Director.

28. Disclaimer

This document and the information contained herein is provided on an "As Is" basis and the OGF disclaims all warranties, express or implied, including but not limited to any warranty that the use of the information herein will not infringe any rights or any implied warranties of merchantability or fitness for a particular purpose.

29. Full Copyright Notice

30. References

[CCSID] Coded Character Set Identifiers (CCSID) http://www-01.ibm.com/software/globalization/ccsid/ccsid_registered.jsp

[XML10] XML 1.0 http://www.w3.org/TR/REC-xml

[XML11] XML 1.1 http://www.w3.org/TR/xml11/

[XMLNS10] Namespaces in XML http://www.w3.org/TR/REC-xml-names/

[XSDLV1] XML Schema Part 1: Structures http://www.w3.org/TR/xmlschema-1/ ,

           XML Schema Part 2: Datatypes http://www.w3.org/TR/xmlschema-2/

[XPath2] XML Path Language (XPath) 2.0 http://www.w3.org/TR/xpath20/

[XMLInfo] XML Information Set (Second Edition) http://www.w3.org/TR/xml-infoset

[Unicode] Unicode (now at version 4.0) http://www.unicode.org/

[ICUDecForm] http://icu.sourceforge.net/apiref/icu4c/classDecimalFormat.html#_details

[ICUCalForm] http://icu.sourceforge.net/apiref/icu4c/classSimpleDateFormat.html and http://userguide.icu-project.org/formatparse/datetime

[IANA] IANA character set encoding names: (http://www.iana.org/assignments/character-sets)

[XMLSch] XML Schema: http://www.w3.org/XML/Schema

[RFC 2119] IETF (Internet Engineering Task Force). *RFC 2119: Key words for use in RFCs to Indicate Requirement Levels*. S. Bradner. 1997.

[OMG] OMG "CAM" TD Model: Object Management Group (OMG) "UML Profile and Interchange Models for Enterprise Application Integration (EAI) Specification" formal/04-03-26, March 2004. Sectioin 7.3.2. Available at http://www.omg.org/cgi-bin/doc?formal/2004-03-26

[XSIL] XSIL homepage, http://www.cacr.caltech.edu/SDA/xsil/

[BFD] Binary Format Description (BFD) Language, http://collaboratory.emsl.pnl.gov/sam/bfd/

[PERLRE] PERL regular expressions. http://perldoc.perl.org/perlre.html#Extended-Patterns

[JAVARE] Java regular expressions. http://download.oracle.com/javase/1.4.2/docs/api/java/util/regex/Pattern.html

[RDP] recursive descent parser**.** A "top-down" parser built from a set of mutually-recursive procedures or a non-recursive equivalent where each such procedure usually implements one of the productions of the grammar. Thus the structure of the resulting program closely mirrors that of the grammar it recognises.
["Recursive Programming Techniques", W.H. Burge, 1975, ISBN 0-201-14450-6].
(1995-04-28)

[OLSON] http://www.iana.org/time-zones

Comment [SMH649]: Add paragraph break

Comment [SMH650]: Add paragraph break

31. Appendix A:Escape Scheme Use Cases

31.1   Escape Character same as dfdl:escapeEscapeCharacater

EscapeKind='escapeCharacter', escapeCharacter='/', ecsapeEscapeCharacater='/', separator=';', extraEscapeCharacter='?'

Logical Data	*Physical Data / Representation*
………………………………..	………………………………..
……………/…………………..	……………//…………………..
……………/…./……………….	……………//…..//……………….
……………//…………………..	……………////…………………..
/…………………………………..	//…………………………………..
………………………………….../	………………………………….//
/……………/…………………..	//……………//…………………..
………………/…………………../	………………//…………………//
…………….;……………………	………………/;……………………
……………../;………………….	……………///;………………….
;…………………………………..	/;…………………………………..
……………?………………….	……………/?………………….

31.2   Escape Character different from dfdl:escapeEscapeCharacater

EscapeKind='escapeCharacter', escapeCharacter='/', ecsapeEscapeCharacater='%', separator=';', extraEscapeCharacter='?'

Logical Data	*Physical Data / Representation*
………………………………	………………………………
……………/…………………..	……………%/…………………..
……………/…./……………….	……………%/….%/…………….
……………//…………………..	……………%/%/…………………..
/…………………………………..	%/…………………………………..
………………………………….../	………………………………….%/
/……………/…………………..	%/……………%/…………………..
………………/…………………../	………………%/………………….%/
…………….;…………………..	………………/;…………………..
……………../;………………….	………………%//;………………..
;…………………………………..	/;…………………………………..
……………?………………….	……………/?………………….
……………%………………..	……………%………………..
……………%/………………..	……………%%/………………..

| …………..…/%……………….. | ………………%/%……………….. |

EscapeKind='escapeCharacter', escapeCharacter='/', ecsapeEscapeCharacater='%',
separator='sep', extraEscapeCharacter='?'

Logical Data	Physical Data / Representation
…………….**sep**………………..	………………/**sep**………………..
………………/**sep**………………..	………………%//**sep**………………..
sep………………………………..	/**sep**………………..………………..

31.3 Escape block with different start and end characters

EscapeKind='escapeBlock', escapeBlockStart='[', escapeBlockEnd=']',
ecsapeEscapeCharacater='%', separator=';', extraEscapeCharacter='?'

Logical Data	Physical Data / Representation
……………………………....	…………….…………………..
[……………………………..	**[[**……………………………..**]**
]……………………………..	**]**……………………………....
……………**[**………………..	……………**[**………………..
……………**]**………………..	……………**]**………………..
…………………………….**]**	…………………………….**]**
[[……………………………..	**[[[**……………………………..**]**
…………………………….**]]**	…………………………….**]]**
……………**[[**………………..	……………**[[**………………..
……………**]]**………………..	……………**]]**………………..
[…………………………….**]**	**[[**……………………………..%**]]**
[……………**]**………………..	**[[**……………%**]**……………….**]**
……………**[**……………….**]**	……………**[**……………….**]**
[…………**[**……………….**]**	**[[**…………….**[**……………….%**]]**
[……………**]**……………….**]**	**[[**……………%**]**……………….%**]]**
[[…………………………….**]**	**[[[**……………………………..%**]]**
[…………………………….**]]**	**[[**……………………………..%**]**%**]]**
[[…………………………….**]]**	**[[[**……………………………..%**]**%**]]**
……………%………………..	……………%………………..
……………%%………………...	……………%%………………...
…………….%**[**………………..	…………….%**[**………………..
…………….%**]**………………..	…………….%**]**………………..
%**[**……………………………..	%**[**……………………………..
…………………………….%**]**	…………………………….%**]**
%**[**…………………………..%**]**	%**[**……………………………..%**]**

[............%........................]	[[............%........................%]]
[............%]........................]	[[............%%%]........................%]]
................;..................	[................;..................]
................%;..................	[................%;..................]
[................;....................]	[[................;....................%]]
................?..................	[................?....................]

31.4 Escape block with same start and end characters

EscapeKind='escapeBlock', escapeBlockStart=' '', escapeBlockEnd=' '', ecsapeEscapeCharacater='%', separator=';', extraEscapeCharacter='?'

Logical Data	*Physical Data / Representation*
....	....
'..................................	'%'................................'
................'....................	................'....................
................................'	................................'
".................................	'%'%'.................................'
................................."	................................."
................"..................	................"..................
'.................................'	'%'................................%'"
'................'..................	'%'................%'...................'
................'................'	................'................'
'................'...................'	'%'................%'...................%'"
"................................'	'%'%'................................%'"
'................................"	'%'................................%'%'"
"................................"	'%'%'................................%'%'"
................%..................	................%..................
................%%..................	................%%..................
................%'..................	................%'..................
%'..................................	%'..................................
...............................%'	...............................%'
'...............................%'	'%'...............................%%'"
%'................................%'	%'................................%'
'............%........................'	'%'............%........................%'"
'............%'........................'	'%'............%%'........................%'"
................;..................	'................;...................'
................%;..................	'................%;...................'

'................;....................'	'%'................;....................%''
.................?...................	'................?...................'

32. Appendix B: Encoding of delimiters different from encoding of data (eg, initiator and terminator different to data)

Use <xs:sequence> to wrap the element and carry the delimiters, for example:

```
<xs:sequence dfdl:encoding="ascii" dfdl:separator=":">
   <xs:sequence dfdl:encoding=" ebcdic-cp-us" dfdl:initiator="VAL"

               dfdl:terminator="END">
     <xs:element name="val" type="..." dfdl:encoding="ascii" />
   </xs:sequence>
</xs:sequence>
```

The same technique can be used with dfdl:ignoreCase when the case-sensitivity of data is different to that of surrounding delimiters.

Comment [m652]: OPEN

Remove – belongs in an advanced tutorial. Is missing caveats about what lengthKinds are sensible in this case, etc.

33. Appendix C: Rationale for Single-Assignment Variables

DFDL is intended to be a description language. That is, the capture of a data format should be as descriptive/declarative as possible.

An additional quite critical goal for DFDL is that it allows very high performance implementations, including use of parallel processing wherever possible.

DFDL contains an expression language with variables for use in creating parameterized DFDL schemas.

However, the way variables can be used in DFDL is quite constrained. Specifically, the variables are single-assignment.

Single-assignment variables solve a number of problems.

First, they keep the schema more declarative, because the name of a variable represents a value, not a location. Before assignment, the value is not yet known, after the assignment the value is known, but the consumer of the value need only know the name, and need not be aware of the mechanism by which it gets its value or when.

Second, single-assignment variables avoid over-constraining the implementation, thereby preserving the potential for high-performance and parallel processing.

Some digression is useful here: Any variable creates a data dependency in order of processing. The part of the schema reading/using the variable's value depends upon the data value coming from the part of the schema providing that value. This kind of data dependency is inherent and inescapable. Values must be created before they can be used.

However, if you consider a variable to be a location that can be assigned repeatedly, then things are more complex because you not only have data dependency on the value (one part of the schema writes the location, another reads that location), but you have the dependency in the other direction: you must read the location before it can be used again for the *next* value. This is usually called anti-dependency. Anti-dependency is the enemy of high-performance and parallel execution. It forces specific and artificial sequential ordering on things that is due to the way variable names are allocated to storage locations.

If variables are single-assignment only, then only data-dependencies exist. Anti-dependencies don't exist, and implementations are free to work in any way consistent with the (inescapable) data dependencies.

(DONE)

2013-05-24 – will keep in mind for possible future clarification. But ok for now.

2013-05-09 – Still open. Model groups don't have nil, empty, normal, or absent reps as those are specifically about elements (above).

Still TBD.

I find this statement confusing

Do we just want to say that the length of a sequence or choice is determined bottom up by considering their content? … ah. I think that is also too simplistic.

SMH: It's trying to capture the case when a model group occurs in the content of a complex element

MJB: Issues: First off, a 'local' sequence or choice as opposed to a group reference to a global one? I don't see any difference in how they are treated for purposes of establishing representation. Are we after a sequence or choice that is not the immediate schema-child of a complexType def?

Second. What are we trying to convey by the phrase "delimiting regime of the parent". The statement needs to cover the case where the parent element isn't delimited at all.

OPEN

Stopvalue related.

Changed wording to be 'contains' to take perspective of containing model group.

Inserted this point based on it coming up below in section on PoU.

2013-05-09: This is about "an occurrence" being known to exist.

Perhaps we need to say that for purposes of known-to-exist, a stop-value because it is a logical value it IS an occurrence of the array. It's just one that we remove later because a stop value is a logical value that is edited out.

(DONE)

I revised this from what it was in the Action 140 materials because I believe it was really wrong, or at least unclear about how discrimination interacts with the known-to-exist/not-exist decision.

SMH: Removed bullet 3 as it is covered by the processing error case.

Decided to keep but change wording slightly

DONE

Moved here from below.

Rationale statement moved to a footnote. Removed redundant is missing or is absent or has no representation. "Is missing" is sufficient.

This statement, however, left me wondering "and how would the DFDL processor determine that?". We don't talk about determining that something is missing. We talk about determining if it is absent rep, which is one way it can be missing, but the other way we don't discuss algorithmically.

It also feels like it is in the wrong place.

Furthermore: this feels odd because the parser algorithm really dosen't determine that things are missing until after a bunch of conditional behaviors associated with complex types. So saying it doesn't recurse when it has established that it is missing seems wrong.

I think this is supposed to say that it will not descend into a complex element when it has established that the element occurrence is empty(such as via lengthKind='prefixed' or expression or pattern. In other cases determining empty however requires that you know if it is zero-length or not, and for some lengthKinds, that requires descending. (implicit, and delimited).