Stephen M Hanson (IBM)

Data Format Description Language (DFDL) v1.0 Experience Document 2

Status of This Document

Grid Working Document (GWD)

Abstract

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

It describes shortcomings experienced in the area of 'missing' elements, default value handling, repeating elements and sequence separator suppression.

All errata have been incorporated into a revised Data Format Description Language (DFDL) 1.0 specification (GFD-P-R.207).

Stephen M Hanson (IBM)

Contents

Stephen M Hanson (IBM)

1. Introduction

DFDL Working Group Action 140 was raised in September 2011 to address shortcomings experienced in the DFDL 1.0 specification in the area of 'missing' elements and default value handling, particularly on parsing. The resultant investigation was wide ranging and uncovered further issues about data representation, repeating elements and sequence separator suppression.

This document records the conclusions of DFDL Working Group Action 140, and should be treated as a companion document to DFDL 1.0 Experience Document 1 [DFDLX1]. Specifically it provides the detailed content for these errata:

- Erratum **3.26**
- Erratum **3.11**
- Erratum **3.14**
- Erratum **2.115**

This document uses terminology defined in [DFDLX1] erratum **2.112**, and refers to the revised grammar in [DFDLX1] Chapter 4.

2. Erratum 3.26. Empty, Missing and Defaults

As specified in the original DFDL 1.0 specification [DFDL], default values are used as follows.

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

The parsing behaviour has the effect of making an invalid data stream valid. This is not actually a good idea. Why is DFDL trying to handle missing required occurrences in a data stream? If an occurrence may be missing from the data stream, it should be modelled as optional. Further this is not how XML Schema 1.0 [XSDL1] uses default values for elements.

For elements, XML Schema 1.0 uses defaults to fill in values for occurrences that are *present but have empty content*. We shall use this principle for DFDL, as the main use case for using defaults on parsing is supplying a value for an empty required occurrence of a simple element (the CSV adjacent separator example). In order for this to work, we must be able to distinguish clearly between an empty occurrence and a missing occurrence when parsing.

Some definitions are needed to cover the range of representations that are possible in the data stream for an element. These definitions assume the revised grammar from Chapter 4 of DFDL 1.0 Experience Document 1 [DFDLX1].

Nil representation
An element occurrence has a nil representation if the element is nillable and the occurrence either:

a) conforms to the grammar for SimpleNilLiteralElementRep or ComplexNilLiteralElementRep. ***NilElementInitiator*** and ***NilElementTerminator*** regions must be conformant with 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 ***NilLogicalElementValue***.

LeadingAlignment, TrailingAlignment, PrefixLength regions may be present.

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. ***EmptyElementInitiator*** and ***EmptyElementTerminator*** regions must be conformant with emptyValueDelimiterPolicy. (If non-conformant it is not a processing error and the representation is not empty). The occurrence's content in the data stream is of length zero. LeadingAlignment, TrailingAlignment, PrefixLength regions may be present.

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.

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. 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 (this is dependent on lengthKind). If this happens, and this length zero representation does not conform to either the nil representation or the empty representation or the normal representation, then it is the absent representation, and it behaves *as if the element occurrence is 'missing'*.

Missing
When parsing, an element occurrence is missing if it does not have any of the above representations, or it has the absent representation. When unparsing, an element occurrence is missing if there is no item in the infoset.

When parsing, an occurrence is 'known to exist' if it has normal, nil or empty representation, or an occurrence is 'known not to exist' if it has absent representation or is missing.

Examples

The following examples illustrate missing and empty.

```
<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.

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

Note that round tripping is not 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 nilValueDelimiterPolicy is the same as emptyValueDelimiterPolicy, then when parsed the Infoset will contain nil.

Establishing representation when parsing

If a processing error or schema definition error occurs either when parsing a simple element, or when parsing a complex element and a processing error is not suppressed by an enclosed point of uncertainty, then the element occurrence is 'known not to exist'. This is equivalent to the element being missing.

If no such error occurs, then an element occurrence either has a representation (one of nil, empty, normal or absent) or is missing.

If it has a representation, then it must be established if it is nil, empty, normal or absent. Key to this is to see if the content is of length zero. This is lengthKind dependent.

- explicit => length is zero (either fixed or from expression evaluation)
- prefixed => prefix length is zero
- implicit (simple) => length is zero from type facets
- implicit (complex) => consumed length is zero upon return from descending into children.
- delimited => length is zero after scanning for delimiter(s)
- pattern => pattern returns zero length match
- endOfParent => already positioned at parent's end so length is zero

For a simple element, length plus initiator and terminator enables the representation to be established.

For a complex element, length plus initiator and terminator enables the nil representation to be established[1], but all other representations can only be determined by descending into the complex type for the element. If the descent returns successfully (that is, no unsuppressed processing error occurs) then the other representations may be established.

The DFDL parser shall not descend into a complex element when it has established that the element occurrence does ***not*** have a representation or is missing or has the absent representation. 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).

For the purposes of establishing representation, a local sequence or choice effectively has lengthKind 'implicit', except that delimiting regime of parent is retained.

Empty representation when parsing

[1] It is a schema definition error if a complex element is nillable 'true' and lengthKind 'implicit'.

If *empty* representation is established when parsing, the possibility of applying a default value arises. Essentially, if a required occurrence of an element has empty representation, then a default value 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 emptyValueDelimiterPolicy for the element, and for a complex element the parser must have descended into the type and returned with no unsuppressed processing error.

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. This behaviour is independent of occursCountKind.

Simple element (non-string)

Required occurrence: If a XSD 'default' or 'fixed' property is specified then an item is added to the Infoset using the value of the property, otherwise nothing is added to the Infoset. (This may cause a subsequent processing error – see 'Required occurrences' below).

Optional occurrence: Nothing is added to the Infoset.

Simple element (string)

Required occurrence: If a XSD 'default' or 'fixed' property is specified then an item is added to the infoset using the value of the property, otherwise an item is added to the Infoset using empty string as the value.

Optional occurrence: If emptyValueDelimiterPolicy is not 'none'[2] then an item is added to the Infoset using empty string as the value, otherwise nothing is added to the Infoset.

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

Complex element

Required occurrence: An item is added to the Infoset.

Optional occurrence: If 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)

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 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

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

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.

Missing when unparsing

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

There are two main cases to consider. This behaviour is independent of occursCountKind.

Simple element

Required occurrence: If a XSD 'default' or 'fixed' property is specified then an item is added to the augmented Infoset using the property value, otherwise nothing is added. (This may cause a subsequent processing error – see 'Required occurrences' below).

Optional occurrence: Nothing is added to the augmented Infoset.

Complex element

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

Optional occurrence: Nothing is added to the augmented Infoset.

For a required occurrence, the unparser descends into the complex type:

- 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 value may give rise to a processing error, as described below.
- For a choice, each branch is examined in schema order and the above rules applied recursively to the branch. The lack of a default value may give rise to a processing error, as described below, 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.

Required occurrences

The specification currently has the concept of 'Required in a required context'. This was added so that the DFDL parser did not cause speculation to succeed by the application of defaults making a bad data stream good. But as we are now saying that the parser does not apply defaults for missing element occurrences, then this concept does not need to be stated explicitly, and the sub-section should be removed.

On parsing, if a required occurrence does not produce an item in the Infoset (after any default is applied) then it is a processing error or a validation error (if enabled), dependent on occursCountKind (see section 3).

On unparsing, if a required occurrence does not produce an item in the augmented Infoset (after any default is applied) then it is a processing error or a validation error (if enabled), dependent on occursCountKind (see section 3).

Optional occurrences

On parsing, nothing is added to the Infoset for an optional occurrence if it is missing or has the absent representation. If it has empty representation, then there are circumstances when an item is added to the Infoset, as described earlier. This is independent of occursCountKind.

On unparsing, nothing is added to the augmented Infoset nor output to the data stream for an optional occurrence if it is missing (including any framing). This is independent of occursCountKind.

3. Erratum 3.11. Arrays

The original DFDL 1.0 specification [DFDL] does not fully define the behaviour for the different occursCountKind property enums. It is especially light on unparsing behaviour. We rectify that here.

Parsing

The full behaviour for parsing arrays and non-arrays is:

```
If minOccurs = maxOccurs = 1
  Expect exactly 1 occurrence
  Processing error if no occurrence found or defaulted
  Stop looking after this occurrence found or defaulted
  occursCountKind is never examined and need not be defined
Else //
  Select occursCountKind
    Case: fixed
      Schema definition error if minOccurs <> maxOccurs
      Expect maxOccurs occurrences
      Processing error if < minOccurs occurrence found or defaulted
      Stop looking when maxOccurs occurrences found
    Case: implicit
      Expect up to maxOccurs occurrences
      Processing error if < minOccurs occurrences found or defaulted
      Stop looking if >= minOccurs occurrences found and known not to
      exist occurs for an occurrence
      Stop looking if and when maxOccurs occurrences found (if not
      unbounded)
    Case: parsed
      Expect any number of occurrences
      Parse as many occurrences as possible until known not to exist
      occurs for an occurrence
      Validation error if < minOccurs occurrences found or defaulted
      Validation error if > maxOccurs occurrences found or defaulted
    Case: expression
      Evaluate occursCount to give number of occurrences
      Expect occursCount occurrences
      Processing error if occursCount occurrences not found
      Stop looking when occursCount occurrences found
      Validation error if < minOccurs occurrences found or defaulted
      Validation error if > maxOccurs occurrences found or defaulted
    Case: stopValue
      Expect any number of occurrences
      Parse occurrences until logical stop value is found
      Processing error if stop value not found even when zero
      occurrences
      Stop value is never added to Infoset
      Validation error if < minOccurs occurrences found or defaulted
      Validation error if > maxOccurs occurrences found or defaulted
Endif
```

A ‘found occurrence’ is one that results in an item being added to the Infoset. Additionally,

the DFDL parser may apply a default value when it encounters an occurrence with an empty representation, as described in section 2.

When parsing an array, points of uncertainty (PoU) only occur for certain occursCountKinds, 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).

Unparsing

The full behaviour for unparsing arrays and non-arrays is:

```
If minOccurs = maxOccurs = 1
  Expect exactly one occurrence
  Processing error if no occurrence found or defaulted
  Processing error if more than 1 occurrence found
  occursCountKind is never examined and need not be defined
Else
  Select occursCountKind
    Case: fixed
      Schema definition error if minOccurs <> maxOccurs
      Expect maxOccurs occurrences
      Processing error if < minOccurs occurrences found or defaulted
      Processing error if > maxOccurs occurrences found
    Case: implicit
      Expect up to maxOccurs occurrences
      Processing error if < minOccurs occurrences found or defaulted
      Processing error if > maxOccurs occurrences found
    Case: parsed, expression
      Expect any number of occurrences
      Validation error if < minOccurs occurrences found or defaulted
      Validation error if > maxOccurs occurrences found
    Case: stopValue
      Expect any number of occurrences
      Logical stop value unparsed and output after last occurrence
      Validation error if < minOccurs occurrences found or defaulted
      Validation error if > maxOccurs occurrences found
Endif
```

A ‘found occurrence’ is one that is in the Infoset. Additionally, the DFDL unparser may apply a default when an occurrence is missing from the Infoset, as described in section 2.

Array and sequence equivalence

The processing of an array is similar to the processing of an equivalent sequence of elements. The following two schemas have the same result assuming any set of identical DFDL element

properties applied to them[3] (excluding necessary name differences due to UPA rules, and any other differences described by the Notes that follow).

```
<xs:sequence>
       <xs:element name="a" type="string" minOccurs ="2" maxOccurs="4"  />
</xs:sequence>

<xs:sequence>
       <xs:element name="a1" type="string" />
       <xs:element name="a2" type="string" />
       <xs:element name="a3" type="string" minOccurs="0" />
       <xs:element name="a4" type="string" minOccurs="0" />
</xs:sequence>
```

Notes:

- The number of elements in the equivalent sequence is maxOccurs, unless occursCountKind is 'expression' in which case the number is occursCount.
- When occursCountKind is 'stopValue' the sequence ends with an additional, hidden, simple element with the same properties, to handle the stop value itself.
- When occursCountKind is 'stopValue', 'parsed' or 'expression' it is a validation error if 'a1' and/or 'a2' are missing from the sequence (rather than a processing error if the first two 'a' occurrences are missing from the array).

Forward progress requirement

It is a processing error when maxOccurs is 'unbounded' and the position in the data does not move during the parsing of an occurrence of the element including any associated separator (that is, the particle for the occurrence). This is to prevent an infinite loop.

Parsing occurrences with non-normal representation

Each time round the array loop, length extraction properties for the element are re-evaluated. It is therefore possible to have occurrences with different representations (nil, empty, normal, absent) in the same array (although with some lengthKinds certain combinations of representations are not possible).

Occurrences with nil representation are added to the Infoset with value 'nil'.

Occurrences with empty representation are either added or not added to the Infoset according to the rules in section 2 above.

Occurrences with absent representation are not added to the Infoset. For a required occurrence it may be a processing error, dependent on occursCountKind.

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'.

1. If the indices of the occurrences are significant and need to be preserved, then the array may be modelled using an element with nillable 'true', nilKind 'literalValue' and nilValue '%ES;'. All

[3] With the exception of properties that are not permitted on arrays, such as inputValueCalc and outputValueCalc

occurrences with empty representation will then produce nil values in the Infoset, so the absolute positions of all occurrences are preserved.

2. If the indices of the occurrences are not significant, then the array should be modelled using an element with 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.

This behaviour is independent of occursCountKind unless explicitly stated otherwise.

4. Erratum 3.14. Separator Suppression Policy

The description in the original DFDL 1.0 specification [DFDL] of DFDL processor behaviour when a sequence has a separator does not provide enough detail. The content is added to section 14.2 and the table in section 14.2.1 is replaced.

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.

These properties combine to define the grammar for a sequence group with sequenceKind 'ordered'. Not all combinations of the properties will give rise to a consistent grammar, 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.

1. 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 separatorSuppressionPolicy to 'required', 'trailingEmptyStrict' or 'trailingEmpty'

2. 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 any optional zero-length occurrence. Speculative parsing and backtracking must be used to identify each occurrence. A non-positional sequence can be modelled by setting separatorSuppressionPolicy to 'anyEmpty'.

separatorSuppressionPolicy	Enum Valid values ’never’, ‘anyEmpty’, ‘trailingEmpty’, ‘trailingEmptyStrict’ Only applicable if separator is not "" (empty string) and sequenceKind is ‘ordered’. Controls the circumstances when separators are expected in the data when parsing, or generated when unparsing, if an element occurrence or group has a representation of length zero. See section **Error! Reference source not found. Error!**

	Reference source not found.. When sequenceKind is 'unordered' then 'anyEmpty' is implied. Annotation: dfdl:sequence, dfdl:group (sequence)

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 the element is required
- The occursCountKind of the element
- The separatorSuppressionPolicy of the sequence
- Whether occurrences are optional or required
- Whether occurrences are trailing
- The representation of the occurrences

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.

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.

Separator suppression policy	**Explanation**
Never	All occurrences MUST be found in the data, along with their associated separator.
trailingEmptyStrict	Trailing occurrences MUST be omitted from the data, along with their associated separator.
trailingEmpty	Trailing occurrences MAY be omitted from the data, along with their associated separator.
anyEmpty	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.

It is a schema definition error if a sequence has separatorSuppressionPolicy 'never' and a child element has occursCountKind 'implicit' and maxOccurs 'unbounded'.

It is a schema definition error if a sequence has separatorSuppressionPolicy 'trailingEmptyStrict' or 'trailingEmpty', and a child element has occursCountKind 'implicit' and 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.

Parsing

When an element is required and is not an array then one occurrence is always expected along with its separator. The separatorSuppressionPolicy is not applicable and the implied behaviour is ‘never’.

Otherwise the behaviour is dependent on occursCountKind.

When occursCountKind is ‘fixed’ minOccurs occurrences are always expected along with their separators. The separatorSuppressionPolicy is not applicable and the implied behaviour is ‘never’.

When occursCountKind is ‘expression’ occursCount occurrences are always expected along with their separators. The separatorSuppressionPolicy is not applicable and the implied behaviour is ‘never’.

When occursCountKind is ‘parsed’ any number of occurrences and their separators are expected. The separatorSuppressionPolicy is not applicable and the implied behaviour is ‘anyEmpty’.

When occursCountKind is ‘stopValue’, any number of occurrences and their separators are expected followed by the stop value and its separator. The separatorSuppressionPolicy is not applicable and the implied behaviour is ‘anyEmpty’.

When occursCountKind is ‘implicit’, between minOccurs and maxOccurs (inclusive) occurrences and their separators are expected. The 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 separatorSuppressionPolicy. The number of occurrences also depends whether 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 separatorPosition.

Note: In the matrices below, it is important that the information is interpreted correctly. The separatorSuppressionPolicy property is carried on the sequence. The occursCountKind property is carried on an *element* in that sequence.

<table>
<tr><th rowspan="4">dfdl:
separatorSuppressionPolicy</th><th colspan="6">dfdl:occursCountKind 'implicit'</th></tr>
<tr><th colspan="4">Potentially Trailing</th><th colspan="2">Not Potentially Trailing</th></tr>
<tr><th colspan="2">maxOccurs unbounded</th><th colspan="2">maxOccurs bounded</th><th rowspan="2">maxOccurs
unbounded</th><th rowspan="2">maxOccurs
bounded</th></tr>
<tr><th>Element not declared last</th><th>Element declared last</th><th>Element declared last or occurrence followed by end-of-group</th><th>Element not declared last and occurrence not followed by end-of-group</th></tr>
<tr><td>never</td><td colspan="2" rowspan="3">Schema definition error</td><td colspan="2" rowspan="2"></td><td rowspan="3">Schema definition error</td><td rowspan="3">RepDef(min) ~ Rep(max - min)</td></tr>
<tr><td>trailingEmptyStrict</td><td>RepDef(min) [~ Rep(M < INF) ~ RepNonZero(1)]</td><td>RepDef(min) [~ Rep(M < max - min) ~ RepNonZero(1)]</td><td rowspan="2">RepDef(min) ~ Rep(max - min)</td></tr>
<tr><td>trailingEmpty</td><td rowspan="2">RepDef(min) ~ Rep(M < INF)</td><td rowspan="2">RepDef(min) ~ Rep(M <= max - min)</td></tr>
<tr><td>anyEmpty</td><td colspan="2"></td><td>RepDef(min) ~ Rep(M < INF)</td><td>RepDef(min) ~ Rep(M <= max - min)</td></tr>
</table>

Terminology used in the matrix:

RepDef(min) means minOccurs occurrences of nil, empty or normal representation[4]. These are required occurrences so default rules apply for empty representations. If permitted, minOccurs may be 0, in which case there are no 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[5]. This is an optional occurrence so default rules do not apply.

Unparsing

When an element is required and is not an array then one occurrence is always output along with its separator. The separatorSuppressionPolicy is not applicable and the implied behaviour is 'never'.

Otherwise the behaviour is dependent on occursCountKind.

When occursCountKind is 'fixed' or 'expression' the occurrences in the augmented Infoset are always output along with their separators. The separatorSuppressionPolicy is not applicable and the implied behaviour is 'never'.

When occursCountKind is 'parsed' non zero-length occurrences in the augmented Infoset are output along with their separators. The separatorSuppressionPolicy is not applicable and the implied behaviour is 'anyEmpty'.

[4] Absent representation implies processing error for 'implicit' when less than or equal to minOccurs.
[5] Absent representation always implies zero-length.

When 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 separatorSuppressionPolicy is not applicable and the implied behaviour is 'anyEmpty'.

When occursCountKind is 'implicit' the occurrences in the augmented Infoset are output along with their separators. The 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 separatorSuppressionPolicy. The number of occurrences also depends whether 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 separatorPosition.

<table>
<tr><th rowspan="4">dfdl:
separatorSuppressionPolicy</th><th colspan="5">dfdl:occursCountKind 'implicit'</th></tr>
<tr><th colspan="4">Potentially Trailing</th><th colspan="2">Not Potentially Trailing</th></tr>
<tr><th colspan="2">maxOccurs unbounded</th><th colspan="2">maxOccurs bounded</th><th rowspan="2">maxOccurs unbounded</th><th rowspan="2">maxOccurs bounded</th></tr>
<tr><th>Element not declared last</th><th>Element declared last</th><th>Element declared last or occurrence followed by end-of-group</th><th>Element not declared last and occurrence not followed by end-of-group</th></tr>
<tr><td>Never</td><td colspan="2" rowspan="3">Schema definition error</td><td colspan="2">Unparse N occurrences ~ unparse (maxOccurs -- N) trailing zero-length occurrences</td><td rowspan="3">Schema definition error</td><td rowspan="3">Unparse N occurrences ~ unparse (maxOccurs -- N) trailing zero-length occurrences</td></tr>
<tr><td>trailingEmptyStrict</td><td colspan="2" rowspan="2">Unparse N occurrences (suppressing trailing zero-length occurrences)</td></tr>
<tr><td>trailingEmpty</td></tr>
<tr><td>anyEmpty</td><td colspan="5">Unparse N occurrences (suppressing any optional zero-length occurrences)</td></tr>
</table>

Terminology used in the matrix:

N is the number of elements in the augmented Infoset, which includes any defaults.

5. Erratum 2.115. Round Trip Ambiguities

This chapter highlights some situations 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 does not matter, the string's characters are output, or if the value is the empty string, zero length content is output. (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. If the nilValue "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 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 optional and not present). 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 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*.

6. Security Considerations

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

No additional security issues have been raised.

7. Contributors

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

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

Tim Kimber,
IBM Software Group,
Hursley,
Winchester,UK

Stephanie Fetzer,
IBM Software Group,
Charlotte, USA

8. 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.

9. 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.

10. Full Copyright Notice

11. References

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

[DFDLR] OGF DFDL 1.0 specification - revised
http://www.ogf.org/documents/GFD.207.pdf/

[DFDLX1] DFDL Experience Document 1
<To be added>

[XSDL1] XML Schema Part 1: structures
http://www.w3.org/TR/xmlschema-1/