BS ISO/IEC 23415:2024

BSI Standards Publication

Information technology — Data Format Description Language (DFDL) v1.0 Specification

Documents
Document Sharing Hub

Full document details: BS ISO/IEC 23415:2024

National foreword

This British Standard is the UK implementation of ISO/IEC 23415:2024.

The UK participation in its preparation was entrusted to Technical Committee ICT/1, Information systems co-ordination.

A list of organizations represented on this committee can be obtained on request to its committee manager.

Contractual and legal considerations

This publication has been prepared in good faith, however no representation, warranty, assurance or undertaking (express or implied) is or will be made, and no responsibility or liability is or will be accepted by BSI in relation to the adequacy, accuracy, completeness or reasonableness of this publication. All and any such responsibility and liability is expressly disclaimed to the full extent permitted by the law.

This publication is provided as is, and is to be used at the recipient's own risk.

The recipient is advised to consider seeking professional guidance with respect to its use of this publication.

This publication is not intended to constitute a contract. Users are responsible for its correct application.

© The British Standards Institution 2024
Published by BSI Standards Limited 2024

ISBN 978 0 539 27866 8

ICS 35.060

Compliance with a British Standard cannot confer immunity from legal obligations.

This British Standard was published under the authority of the Standards Policy and Strategy Committee on 30 April 2024.

Amendments/corrigenda issued since publication

Date	Text affected

Documents
Document Sharing Hub

Full document details: BS ISO/IEC 23415:2024

International Standard

ISO/IEC 23415

Information technology — Data Format Description Language (DFDL) v1.0 Specification

First edition
2024-04

Reference number
ISO/IEC 23415:2024(en)

© ISO/IEC 2024

Documents
Document Sharing Hub

Full document details: BS ISO/IEC 23415:2024

COPYRIGHT PROTECTED DOCUMENT

© ISO/IEC 2024

All rights reserved. Unless otherwise specified, or required in the context of its implementation, no part of this publication may be reproduced or utilized otherwise in any form or by any means, electronic or mechanical, including photocopying, or posting on the internet or an intranet, without prior written permission. Permission can be requested from either ISO at the address below or ISO's member body in the country of the requester.

ISO copyright office
CP 401 • Ch. de Blandonnet 8
CH-1214 Vernier, Geneva
Phone: +41 22 749 01 11
Email: copyright@iso.org
Website: www.iso.org

Published in Switzerland

Documents
Document Sharing Hub

Full document details: BS ISO/IEC 23415:2024

Contents

1 Introduction ... 9
1.1 Why is DFDL Needed? ... 10
1.2 What is DFDL? ... 10
1.2.1 Simple Example ... 10
1.3 What DFDL is not ... 13
1.4 Scope of version 1.0 ... 13
2 Overview of the Specification ... 15
3 Notational and Definitional Conventions ... 16
3.1 Glossary and Terminology ... 16
3.2 Failure Types ... 16
4 The DFDL Information Set (Infoset) ... 17
4.1 "No Value" ... 18
4.2 Information Items ... 18
4.2.1 Document Information Item ... 18
4.2.2 Element Information Items ... 18
4.3 DFDL Information Item Order ... 19
4.4 DFDL Augmented Infoset ... 19
5 DFDL Schema Component Model ... 20
5.1 DFDL Simple Types ... 20
5.2 DFDL Subset of XML Schema ... 21
5.3 XSD Facets, min/maxOccurs, default, and fixed ... 22
5.3.1 MinOccurs, MaxOccurs ... 23
5.3.2 MinLength, MaxLength ... 23
5.3.3 MaxInclusive, MaxExclusive, MinExclusive, MinInclusive, TotalDigits, FractionDigits ... 23
5.3.4 Pattern ... 23
5.3.5 Enumeration Values ... 23
5.3.6 Default ... 23
5.3.7 Fixed ... 24
5.4 Compatibility with Other Annotation Language Schemas ... 24
6 DFDL Syntax Basics ... 25
6.1 Namespaces ... 25
6.2 The DFDL Annotation Elements ... 25
6.3 DFDL Properties ... 26
6.3.1 DFDL String Literals ... 28
6.3.2 DFDL Expressions ... 32
6.3.3 DFDL Regular Expressions ... 32
6.3.4 Enumerations in DFDL ... 32
7 Syntax of DFDL Annotation Elements ... 33
7.1 Component Format Annotations ... 33
7.1.1 Property Binding Syntax ... 34
7.1.2 Empty String as a Representation Property Value ... 35
7.2 dfdl:defineFormat - Reusable Data Format Definitions ... 36
7.2.1 Using/Referencing a Named Format Definition: The dfdl:ref Property ... 36
7.2.2 Inheritance for dfdl:defineFormat ... 36
7.3 The dfdl:defineEscapeScheme Defining Annotation Element ... 37
7.3.1 Using/Referencing a Named escapeScheme Definition ... 37
7.4 The dfdl:escapeScheme Annotation Element ... 37

© ISO/IEC 2024 – All rights reserved

Documents
Document Sharing Hub
Full document details: BS ISO/IEC 23415:2024

7.5 The dfdl:assert Statement Annotation Element ... 38
7.5.1 Properties for dfdl:assert ... 38
7.6 The dfdl:discriminator Statement Annotation Element ... 40
7.6.1 Properties for dfdl:discriminator ... 40
7.7 DFDL Variable Annotations ... 43
7.7.1 dfdl:defineVariable Annotation Element ... 43
7.7.2 The dfdl:newVariableInstance Statement Annotation Element ... 44
7.7.3 The dfdl:setVariable Statement Annotation Element ... 45
8 Property Scoping and DFDL Schema Checking ... 47
8.1 Property Scoping ... 47
8.1.1 Property Scoping Rules ... 47
8.1.2 Providing Defaults for DFDL properties ... 47
8.1.3 Combining DFDL Representation Properties from a dfdl:defineFormat ... 48
8.1.4 Combining DFDL Properties from References ... 48
8.2 DFDL Schema Checking ... 51
8.2.1 Schema Component Constraint: Unique Particle Attribution ... 51
8.2.2 Optional Checks and Warnings ... 51
9 DFDL Processing Introduction ... 53
9.1 Parser Overview ... 53
9.1.1 Points of Uncertainty ... 53
9.1.2 Processing Error ... 54
9.1.3 Recoverable Error ... 54
9.2 DFDL Data Syntax Grammar ... 54
9.2.1 Nil Representation ... 56
9.2.2 Empty Representation ... 56
9.2.3 Normal Representation ... 57
9.2.4 Absent Representation ... 57
9.2.5 Zero-length Representation ... 57
9.2.6 Missing ... 57
9.2.7 Examples of Missing and Empty Representation ... 58
9.2.8 Round Trip Ambiguities ... 58
9.3 Parsing Algorithm ... 58
9.3.1 Known-to-exist and Known-not-to-exist ... 59
9.3.2 Establishing Representation ... 60
9.3.3 Resolving Points of Uncertainty ... 61
9.4 Element Defaults ... 62
9.4.1 Definitions ... 62
9.4.2 Element Defaults When Parsing ... 62
9.4.3 Element Defaults When Unparsing ... 64
9.5 Evaluation Order for Statement Annotations ... 65
9.5.1 Asserts and Discriminators with testKind 'expression' ... 66
9.5.2 Discriminators with testKind 'expression' ... 66
9.5.3 Elements and setVariable ... 66
9.5.4 Controlling the Order of Statement Evaluation ... 66
9.6 Validation ... 66
9.7 Unparser Infoset Augmentation Algorithm ... 67
10 Overview: Representation Properties and their Format Semantics ... 68

© ISO/IEC 2024 – All rights reserved

Documents
Document Sharing Hub
Full document details: BS ISO/IEC 23415:2024

11 Properties Common to both Content and Framing 69
11.1 Unicode Byte Order Mark (BOM) 71
11.2 Character Encoding and Decoding Errors 71
11.2.1 Property dfdl:encodingErrorPolicy 72
11.2.2 Unicode UTF-16 Decoding/Encoding Non-Errors 73
11.2.3 Preserving Data Containing Decoding Errors 73
11.3 Byte Order and Bit Order 73
11.4 dfdl:bitOrder Example 73
11.4.1 Example Using Right-to-Left Display for 'leastSignificantBitFirst' 74
11.4.2 dfdl:bitOrder and Grammar Regions 74
12 Framing 75
12.1 Aligned Data 75
12.1.1 Implicit Alignment 76
12.1.2 Mandatory Alignment for Textual Data 76
12.1.3 Mandatory Alignment for Packed Decimal Data 77
12.1.4 Example: AlignmentFill 77
12.2 Properties for Specifying Delimiters 78
12.3 Properties for Specifying Lengths 81
12.3.1 dfdl:lengthKind 'explicit' 82
12.3.2 dfdl:lengthKind 'delimited' 82
12.3.3 dfdl:lengthKind 'implicit' 83
12.3.4 dfdl:lengthKind 'prefixed' 84
12.3.5 dfdl:lengthKind 'pattern' 87
12.3.6 dfdl:lengthKind 'endOfParent' 87
12.3.7 Elements of Specified Length 89
13 Simple Types 92
13.1 Properties Common to All Simple Types 92
13.2 Properties Common to All Simple Types with Text representation 93
13.2.1 The dfdl:escapeScheme Properties 94
13.3 Properties for Bidirectional support for All Simple Types with Text representation 97
13.4 Properties Specific to String 97
13.5 Properties Specific to Number with Text or Binary Representation 99
13.6 Properties Specific to Number with Text Representation 99
13.6.1 The dfdl:textNumberPattern Property 106
13.6.2 Converting logical numbers to/from text representation 110
13.7 Properties Specific to Number with Binary Representation 112
13.7.1 Converting Logical Numbers to/from Binary Representation 113
13.8 Properties Specific to Float/Double with Binary Representation 118
13.9 Properties Specific to Boolean with Text Representation 118
13.10 Properties Specific to Boolean with Binary Representation 119
13.11 Properties Specific to Calendar with Text or Binary Representation 120
13.11.1 The dfdl:calendarPattern property 122
13.11.2 The dfdl:calendarCheckPolicy Property 125
13.12 Properties Specific to Calendar with Text Representation 125
13.13 Properties Specific to Calendar with Binary Representation 126
13.14 Properties Specific to Opaque Types (xs:hexBinary) 127
13.15 Nil Value Processing 127
13.16 Properties for Nillable Elements 127

© ISO/IEC 2024 – All rights reserved

Documents
Document Sharing Hub
Full document details: BS ISO/IEC 23415:2024

14 Sequence Groups ... 131
14.1 Empty Sequences ... 131
14.2 Sequence Groups with Separators ... 132
14.2.1 Separators and Suppression ... 133
14.2.2 Parsing Sequence Groups with Separators ... 134
14.2.3 Unparsing Sequence Groups with Separators ... 136
14.3 Unordered Sequence Groups ... 138
14.3.1 Restrictions for Unordered Sequences ... 138
14.3.2 Parsing an Unordered Sequence ... 138
14.3.3 Unparsing an Unordered Sequence ... 140
14.4 Floating Elements ... 140
14.5 Hidden Groups ... 141
15 Choice Groups ... 143
15.1 Resolving Choices ... 144
15.1.1 Resolving Choices via Speculation ... 144
15.1.2 Resolving Choices via Direct Dispatch ... 145
15.1.3 Unparsing Choices ... 145
16 Properties for Array Elements and Optional Elements ... 146
16.1 The dfdl:occursCountKind property ... 146
16.1.1 dfdl:occursCountKind 'fixed' ... 146
16.1.2 dfdl:occursCountKind 'implicit' ... 147
16.1.3 dfdl:occursCountKind 'parsed' ... 147
16.1.4 dfdl:occursCountKind 'expression' ... 147
16.1.5 dfdl:occursCountKind 'stopValue' ... 147
16.2 Default Values for Arrays ... 148
16.3 Arrays with DFDL Expressions ... 148
16.4 Points of Uncertainty ... 148
16.5 Arrays and Sequences ... 148
16.6 Forward Progress Requirement ... 148
16.7 Parsing Occurrences with Non-Normal Representation ... 149
16.8 Sparse Arrays ... 149
17 Calculated Value Properties ... 150
17.1 Example: 2d Nested Array ... 151
17.2 Example: Three-Byte Date ... 151
18 DFDL Expression Language ... 154
18.1 Expression Language Data Model ... 154
18.2 Variables ... 155
18.2.1 Rewinding of Variable Memory State ... 155
18.2.2 Variable Memory State Transitions ... 155
18.3 General Syntax ... 156
18.4 DFDL Expression Syntax ... 157
18.5 Constructors, Functions and Operators ... 158
18.5.1 Constructor Functions for XML Schema Built-in Types ... 158
18.5.2 Standard XPath Functions ... 159
18.5.3 DFDL Functions ... 162
18.5.4 DFDL Constructor Functions ... 164
18.5.5 Miscellaneous Functions ... 165
18.6 Unparsing and Circular Expression Deadlock Errors ... 166

© ISO/IEC 2024 – All rights reserved

Documents
Document Sharing Hub
Full document details: BS ISO/IEC 23415:2024

19 DFDL Regular Expressions ... 167
20 External Control of the DFDL Processor ... 168
21 Built-in Specifications ... 169
22 Conformance ... 170
23 Optional DFDL Features ... 171
24 Security Considerations ... 173
25 Authors and Contributors ... 174
26 Intellectual Property Statement ... 175
27 Disclaimer ... 176
28 Full Copyright Notice ... 177
29 References ... 178
30 Appendix A: Escape Scheme Use Cases ... 181
30.1 Escape Character Same as dfdl:escapeEscapeCharacter ... 181
30.2 Escape Character Different from dfdl:escapeEscapeCharacter ... 181
30.2.1 Example 1 - Separator ';' ... 181
30.2.2 Example 2 - Separator 'sep' ... 182
30.3 Escape Block with Different Start and End Characters ... 182
30.4 Escape Block with Same Start and End Characters ... 183
31 Appendix B: Rationale for Single-Assignment Variables ... 185
32 Appendix C: Processing of DFDL String literals ... 186
32.1 Interpreting a DFDL String Literal ... 186
32.2 Recognizing a DFDL String Literal ... 186
32.3 Recognizing DFDL String Literal Part ... 186
33 Appendix D: DFDL Standard Encodings ... 188
33.1 Purpose ... 188
33.2 Conventions ... 188
33.3 Specification Template ... 188
33.4 Encoding X-DFDL-US-ASCII-7-BIT-PACKED ... 188
33.4.1 Name ... 188
33.4.2 Translation table ... 188
33.4.3 Width ... 188
33.4.4 Alignment ... 189
33.4.5 Byte Order ... 189
33.4.6 Example 1 ... 189
33.4.7 Example 2 ... 189
33.5 Encoding X-DFDL-US-ASCII-6-BIT-PACKED ... 191
33.5.1 Name ... 191
33.5.2 Translation Table ... 191
33.5.3 Width ... 192
33.5.4 Alignment ... 192
33.5.5 ByteOrder ... 192
33.5.6 Example 1 ... 192
33.6 References for Appendix D ... 193
34 Appendix E: Glossary of Terms ... 194
35 Appendix F: Specific Errors Classified ... 200
36 Appendix G: Property Precedence ... 202
36.1 Parsing ... 202
36.1.1 dfdl:element (simple) and dfdl:simpleType ... 202

© ISO/IEC 2024 – All rights reserved

Documents
Document Sharing Hub
Full document details: BS ISO/IEC 23415:2024

36.1.2 dfdl:element (complex) ... 205
36.1.3 dfdl:sequence and dfdl:group (when reference is to a sequence) ... 206
36.1.4 dfdl:choice and dfdl:group (when reference is to a choice) ... 207
36.2 Unparsing ... 208
36.2.1 dfdl:element (simple) and dfdl:simpleType ... 208
36.2.2 dfdl:element (complex) ... 212
36.2.3 dfdl:sequence and dfdl:group (when reference is a sequence) ... 213
36.2.4 dfdl:choice and dfdl:group (when reference is a choice) ... 213

© ISO/IEC 2024 – All rights reserved

Documents
Document Sharing Hub
Full document details: BS ISO/IEC 23415:2024

1 Introduction

Data interchange is critically important for most computing. Grid computing, Cloud 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.

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

1. Textual data defined by a format specific schema such as XML[XML] or JSON[JSON].
2. Binary data in standard formats.
3. Data with DFDL descriptors.

Textual XML and JSON data are the most successful data interchange standards to date. All such data are by definition new, meaning created in the Internet era. Because of the large overhead that textual tagging imposes, there is often a need to compress and decompress XML and JSON data. However, there is a high cost for compression and decompression that is unacceptable to some applications. Standardized binary data formats are also relatively new and are 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[1] [ASN1], XDR [XDR], Thrift [Thrift], Avro [AVRO], and Google Protocol Buffers [GPB]. These techniques lack the self-describing nature of XML or JSON data. Scientific formats, such as NetCDF[NetCDF] and HDF[HDF] are used by some communities to provide self-describing binary data. There are also standardized binary-encoded XML data formats such as EXI [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 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[2]. 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 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 ensure 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 intent of DFDL to support canonical data descriptions that correspond closely to the original in-memory representation of the data, and to provide sufficient information to write as well as to read the given format.

---

[1] ASN.1 with any of the prescribed encoding rules: Basic Encoding Rules (BER), Distinguished Encoding Rules (DER), Canonical Encoding Rules (CER) [ASN1CER] or Packed Encoding Rules (PER) [ASN1PER]

[2] Additional examples of descriptive approaches: ASN1 Encoding Control Notation (also known as ITU-T X.692) [ASN1ECN], BFD: Binary Format Description (BFD) Language [BFD]. The largest set of examples of descriptive approaches are all the various proprietary ad-hoc format description languages found almost universally in every commercial database, analytical, or enterprise software system that must take in data.

© ISO/IEC 2024 – All rights reserved

Documents
Document Sharing Hub
Full document details: BS ISO/IEC 23415:2024

1.1 Why is DFDL Needed?

In an era when there are so many standard data formats available the question arises of why DFDL is needed. Ultimately, it is because data formats are rarely a primary consideration when programs are initially created.

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 often harder than just using the native I/O libraries of a programming language.

In time, however, if a software program becomes important either because many people are using it, or it has become important for business or organizational needs, 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.

Indeed, 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.

There are, of course, efforts to smoothly integrate standardized data-format handling into programming languages. However, the above phenomena are not going away any time soon and there is a critical role for DFDL since it allows after-the-fact description of evolving data formats.

1.2 What is DFDL?

DFDL is a language for describing data formats. A DFDL description enables *parsing*, that is, it allows data to be read from its native format and presented as a data structure called the *DFDL Information Set or DFDL Infoset*. This information set describes the common characteristics of parsed data that are required of all DFDL implementations and it is fully defined in Section 4. DFDL implementations MAY provide API access to the Infoset as well as conversion of the Infoset into concrete representations such as XML text, binary XML [EXI] , or JSON [JSON]. DFDL also enables *unparsing*[3]*, that is,* allows data to be taken from an instance of a DFDL information set and written out to its native format.

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

An XML schema is written for the logical model of the data. The schema is augmented with special DFDL annotations and the annotated schema is called a *DFDL Schema*. The annotations are used to describe the native representation of the data.

This approach of extending XSD with format annotations has been extensively used in commercial systems that predate DFDL. The contribution of DFDL for data parsing is creation of a standard for these annotations that is open, comprehensive, and vendor neutral. For unparsing DFDL does more to advance the state of the art by providing some capabilities to automatically compute fields that depend on the length or presence of other data. Prior-generation data format technologies left this difficult task up to application logic to compute.

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 a 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>
```

[3] DFDL uses the term ‘unparsing’ for symmetry with parsing. This is roughly equivalent to the terms ‘marshalling’ or ‘serialization’, but those terms both connote a sequencing order that DFDL does not impose for all formats, so DFDL usesits own distinct term.

© ISO/IEC 2024 – All rights reserved

Documents
Document Sharing Hub
Full document details: BS ISO/IEC 23415:2024

```
  </xs:complexType>
```

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

```
0000 0005 0077 9e8c
169a 54dd 0a1b 4a3f
ce29 46f6
```

To describe the same information in DFDL, the original XML schema document that described the data model is annotated (on the type definition) as follows:

```
  <xs:complexType>
    <xs:sequence>
      <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 is big endian. This is all that a DFDL parser needs to read the data.

In the above, there is a standard XML schema annotation structure:

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

This encapsulates DFDL *annotation elements*. The source attribute on the xs:appinfo element indicates that the annotation is specifically a DFDL annotation.

© ISO/IEC 2024 – All rights reserved

Full document details: BS ISO/IEC 23415:2024

Inside the xs:appinfo there is a single DFDL *format annotation*:

```
            <dfdl:element representation="binary"
                    byteOrder="bigEndian"
                    lengthKind="implicit"
                    binaryFloatRep="ieee" />
```

Within the above annotation element, each attribute is a DFDL *property*, and each property-value pair is called a *property binding*. In the above the attribute 'representation' is a DFDL property name. Here the dfdl:element is a DFDL format annotation and the properties in it are generally called DFDL *representation properties*.

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

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

Once again, the same data model can be annotated, 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" 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"
                        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"
                        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"
                        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"
                        textStandardDecimalSeparator="."
                        lengthKind="delimited"/>
          </xs:appinfo>
```

© ISO/IEC 2024 – All rights reserved

Documents
Document Sharing Hub
Full document details: BS ISO/IEC 23415:2024

```
        </xs:annotation>
      </xs:element>
    </xs:sequence>
  </xs:complexType>
```

Many properties are repeatedly expressed in the example for the sake of simplicity. Later sections of this specification define the mechanisms DFDL provides to avoid this repetition.

1.3 What DFDL is not

DFDL maps data from a native textual or binary 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 DFDL does not intend to provide a mechanism to map data to arbitrary XML models. There are specific limitations on the data models that DFDL can work to:

1. DFDL uses a subset of XML Schema; in particular, XML attributes cannot be used 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.
3. Recursive definitions are not supported.

Point (2) 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, one does not get to design an XML schema to one's preference and then populate it from data. That would involve two steps: first describing the data format and second describing a transformation for mapping it to the structure of the XML schema. DFDL is only about the format part of this problem. There are other languages, such as XSLT [XSLT], which are for transformation. In DFDL,one describes only the format of the data, and the format constrains the nature of the XML schema one must use in its description.

DFDL is also not intended for describing generic formats like XML or JSON (for which schema-aware parsers exist), nor for prescriptive formats like Google Protocol Buffers [GPB] where the format is never exposed and access is via software libraries.

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 XSD 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 XSD 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 groups of elements – A description of an intermediate representation the corresponding Infoset items of which are not exposed in the final Infoset.
g) Basic arithmetic 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 can describe binary data as found in the data structures of COBOL, C, PL1, Fortran, etc., as well as standard binary data in formats like ISO8583 [ISO8583]. DFDL can describe repeating sub-arrays where the length of an array is stored in another location of the structure.

© ISO/IEC 2024 – All rights reserved

Full document details: BS ISO/IEC 23415:2024

DFDL can describe a wide variety of textual data formats such as HL7, X12, CSV, and SWIFT MT [DFDLSchemas]. 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 the combination results in a working format.

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

- Extensibility: There are real examples of proprietary data format description languages that were used as the base of experience from which standard DFDL was derived. 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.

© ISO/IEC 2024 – All rights reserved

Documents
Document Sharing Hub
Full document details: BS ISO/IEC 23415:2024

2 Overview of the Specification

The sections of the specification are

- Section 3, Notational and Definitional Conventions - provides definitions used throughout the specification. Note that terminology is defined at point of first use, but there is a complete Glossary in Appendix E: Glossary of Terms.
- Section 4, The DFDL Information Set (Infoset) - describes the abstract data structure produced by parsing data using a DFDL processor, and which is consumed by a DFDL processor when unparsing data. DFDL contains an expression language, and it is this data structure that the expression language operates on.
- Section 5, DFDL Schema Component Model describes the components that makes up a DFDL schema, and the subset of XML Schema that is used to express them.
- Sections 6, DFDL Syntax Basics and 7, Syntax of DFDL Annotation Elements - describes the syntactic structure of DFDL annotations and introduces the purposes of the various annotations.
- Section 8, Property Scoping and DFDL Schema Checking describes the way DFDL annotations that provide format properties are combined across the parts of the DFDL schema, and also describes static checking that is done on the DFDL schema.
- Section 9, DFDL Processing Introduction covers processing, including the core algorithms for parsing and unparsing data, as well as validation. It introduces the DFDL Data Syntax Grammar, which captures the structure of data that can be described with DFDL, and which is referenced throughout the rest of the specification.
- Section 10, Overview: Representation Properties and their Format Semantics provides an overview of, and Sections 11 to 17 describe in detail, all the DFDL properties. The properties are organized as follows:
    - Common to both Content and Framing (see Section 11)
    - Common Framing, Position, and Length (see Section 12)
    - Simple Type Content (see Section 13) - This is the largest section as it covers properties for all the various simple types, starting with properties that apply to all simple types, then properties for all types with textual representation, and then proceeding through the types, covering textual and binary format properties for each type.
    - Sequence Groups (see Section 14)
    - Choice Groups (see Section 15)
    - Array (i.e., recurring) elements and optional elements (see Section 16)
    - Calculated Values (see Section 17)
- Section 18, DFDL Expression Language covers the XPath-derived expression language that is embedded in DFDL and is used for computing the values of many properties dynamically, as well as for calculated value elements, and assertion checking.
- Section 19, DFDL Regular Expressions, covers the regular expression language used when parsing to isolate elements within the data stream, as well as to check assertions.

The remaining sections and appendices supply additional details of particular importance to implementors of DFDL, or they provide detail and reference material and are referenced from other parts of the specification.

© ISO/IEC 2024 – All rights reserved

Documents
Document Sharing Hub
Full document details: BS ISO/IEC 23415:2024

3 Notational and Definitional Conventions

Examples of DFDL schemas provided herein are for illustration purposes only and for clarity they often do not include all the necessary DFDL properties that would be needed for a complete functional DFDL schema.

3.1 Glossary and Terminology

This specification provides definitions of the terms it uses at the point of first use. However, as this specification will not generally be read linearly, but out of order, a complete glossary is provided in Appendix E: Glossary of Terms.

The capitalized key words MUST, MUST NOT, SHALL, SHALL NOT, SHOULD, SHOULD NOT, MAY, REQUIRED, OPTIONAL, and RECOMMENDED in this document are to be interpreted as described in [RFC2119]. Such usage in capital letters is generally about DFDL implementations and their common or distinguishing characteristics.

When describing requirements for correct usage of the DFDL language by a DFDL Schema author, these same words are used, but are not capitalized. For example, the specification may state "The DFDL fillByte property *must* be a single byte or single character." What is intended by "*must*" here is that if the value for that property does not conform, that it is a Schema Definition Error by the schema author.

Similarly, when describing characteristics of data being parsed or being unparsed, and whether that data conforms to the format described by a DFDL schema, these same words may be used. For example,the specification may state "The representation *must* be followed by a terminating delimiter.", but what is intended by "*must*" in this case is that the consequence of the data not having that terminating delimiter is a Processing Error because the data does not comply with its format specification.

When describing data, the uncapitalized terms *required* and *optional* in this document have specific formal meanings (introduced in Section 5.3.1,MinOccurs, MaxOccurs) having to do with the way element declarations are annotated in the DFDL language. The data corresponding to such an element declaration is also said to be either required or optional, and the DFDL element declaration is said to be for a required element, or an optional element.

3.2 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. These are:

- ***Schema Definition Error*** or ***SDE*** for short - these indicate the DFDL schema is not meaningful. They are generally fatal errors that prevent or stop processing of data.
- ***Processing Error*** - These are errors that occur when parsing or unparsing.
    - At parse time, Processing Errors can cause the parser to search (such as via backtracking) for alternative ways to parse the data as are allowed by the DFDL schema. In that sense parse-time Processing Errors guide the parsing, and when the parser finds an alternative way to parse the data, a prior parse error is said to have been *suppressed*. A parse error that is not suppressed MUST terminate parsing with a diagnostic message.
    - At unparse-time, Processing Errors are generally fatal. They MUST cause unparsing to stop with a diagnostic message.
- ***Validation Error*** - These are errors when optional validation checking is available and enabled. Validation Errors MUST not stop, nor influence, parsing or unparsing behavior. Validation Errors are effectively warnings indicating lack of conformance of the parser output, or the unparser input, with the XML Schema facet constraints, or the XSD maxOccurs and XSD minOccurs values.
- ***Recoverable Error*** - In addition to using XML Schema validation, DFDL also provides the ability to add Recoverable Error assertions to a DFDL schema. These cause diagnostic messages to be created but MUST not stop, nor influence, parsing or unparsing behavior.

© ISO/IEC 2024 – All rights reserved

Documents
Document Sharing Hub
Full document details: BS ISO/IEC 23415:2024

4 The DFDL Information Set (Infoset)

This section defines an abstract data set called the ***DFDL Information Set*** (***Infoset***). Its purpose is to define what is 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 enables unparsing the Infoset and reparsing the resultant data stream to produce the same Infoset.

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

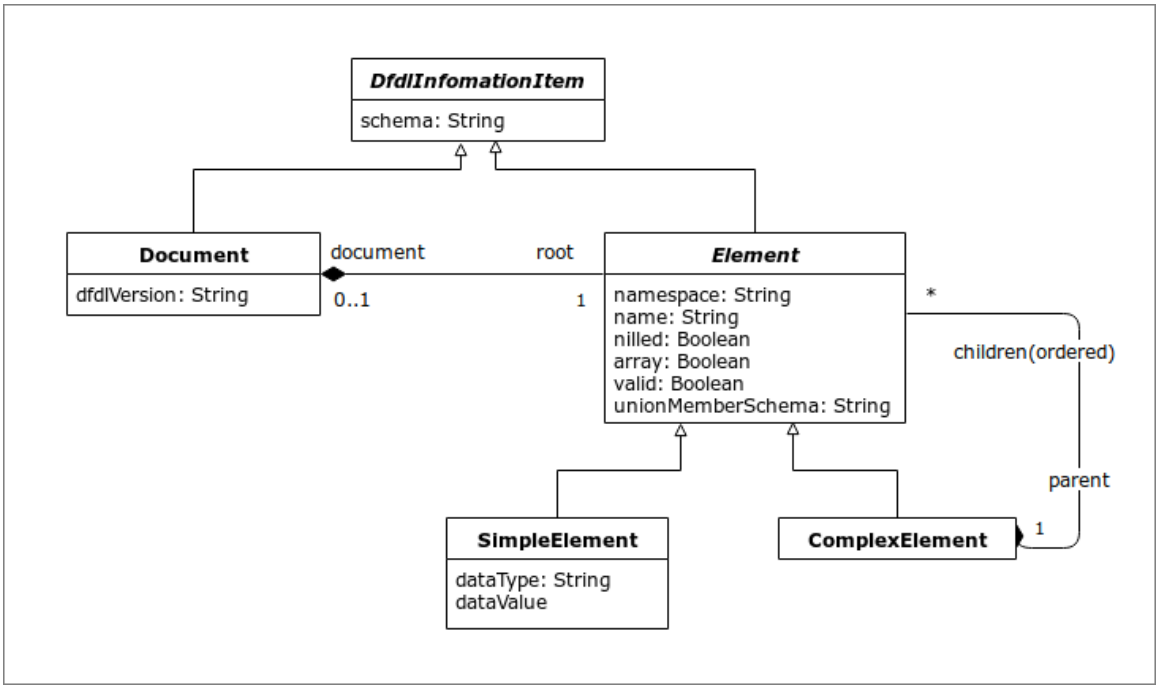

Figure 1 DFDL Infoset Object Model

The DFDL information set is presented above in Figure 1 DFDL Infoset Object Model as an object model using a Unified Modeling Language (UML) class diagram [UML].

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

DFDL describes 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.

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 contains 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 4.2 Information Items.

The DFDL Information Set does not require or favor a specific implementation interface paradigm. 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.

© ISO/IEC 2024 – All rights reserved

Documents
Document Sharing Hub
Full document details: BS ISO/IEC 23415:2024

The DFDL Information Set is similar in purpose to the XML Information Set [XMLInfoset], however, it is not identical, nor a perfect subset, as there are important differences such as that the DFDL Infoset does not have 'text' nodes that are a primary feature of the XML Infoset, as well as that the contents of strings is much less restricted in the DFDL Infoset.

The DFDL Information Set does not have any specific support for comments. When a data format allows for textual data mixed with a comment syntax, then both that data and the content of the comments correspond to DFDL Information Items.

4.1 "No Value"

In the discussion of Information Items and their members below, 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. The concept of no-value is also orthogonal to how nillable elements are represented in the Infoset, which uses a separate **[nilled]** boolean flag, not a distinguished value.

4.2 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]**.

4.2.1 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:

- **[root]** The element information item corresponding to the root element declaration of the DFDL Schema.
- **[dfdlVersion]** String. The version of the DFDL specification to which this information set conforms. For DFDL V1.0 this is 'dfdl-1.0'
- **[schema]** String. This member is reserved for future use.

4.2.2 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.

The [root] member of the document information item corresponds 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:

- **[array]** Boolean. True if the item is an array, meaning that it corresponds to an element having maxOccurs value greater than 1, or 'unbounded'.
- **[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. If the **[nilled]** member is true, then this member has no value.
- **[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 Simple Types.
- **[dataValue]** member has no value, and for a complex element the **[children]** member has no value. If this member is true, then the Infoset item is said to be nil or nilled.
- **[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.
- **[name]** String. The local part of the element name.

© ISO/IEC 2024 – All rights reserved

Documents
Document Sharing Hub
Full document details: BS ISO/IEC 23415:2024

- **[namespace]** String. The namespace, if any, of the element. If the element does not belong to a namespace, the value is the empty string.
- **[nilled]** Boolean. True if the nillable item is nil. False if the nillable item is not nil. If the element is not nillable this member has no value. If this member is true then for a simple element the
- **[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.
- **[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].
- **[unionMemberSchema]**[4] 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.
- **[valid]** Boolean[5]. 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.

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.

4.3 DFDL Information Item Order

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

4.4 DFDL Augmented Infoset

When unparsing, one begins with the DFDL schema and conceptually with the logical Infoset. This Infoset can be sparsely populated because the DFDL Schema can describe default values and computations to be done to obtain the values of some elements. As unparsing progresses and fills in these defaultable and calculated elements, these new item values augment the Infoset, that is, make it bigger. The resulting Infoset is called the *augmented Infoset*. The details of this augmentation process are described in Section 9.7 Unparser Infoset Augmentation Algorithm.

[4] Also, to support PSVI [PSVI] construction.

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

© ISO/IEC 2024 – All rights reserved

Documents
Document Sharing Hub
Full document details: BS ISO/IEC 23415:2024

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.

The shaded boxes have direct corresponding XML Schema syntax and therefore appear in DFDL schema. The unshaded boxes are conceptual classes often used in discussion of DFDL schemas. For example, the ModelGroup class is a generalization of Sequence and Choice classes which are the concrete classes corresponding to xs:sequence and xs:choice constructs of the schema. The class Term is a further generalization encompassing not only ModelGroup, but GroupReference, ElementReference, and ElementDeclaration.

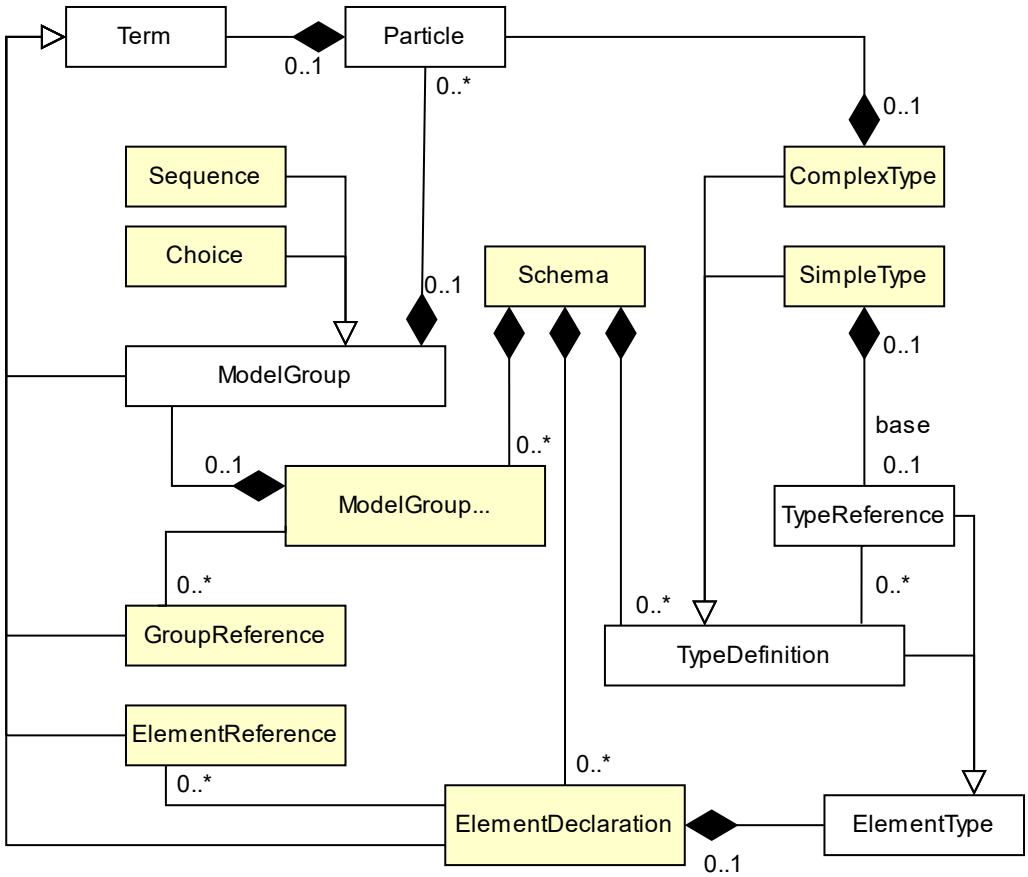

Figure 2 DFDL Schema UML diagram

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

The DFDL Schema Model is expressed using a subset of the XML Schema Description Language (XSD). XSD 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 in general since the DFDL Schema Model does not include many concepts that appear in XML schema.

5.1 DFDL Simple Types

The DFDL 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.

© ISO/IEC 2024 – All rights reserved

Documents
Document Sharing Hub
Full document details: BS ISO/IEC 23415:2024

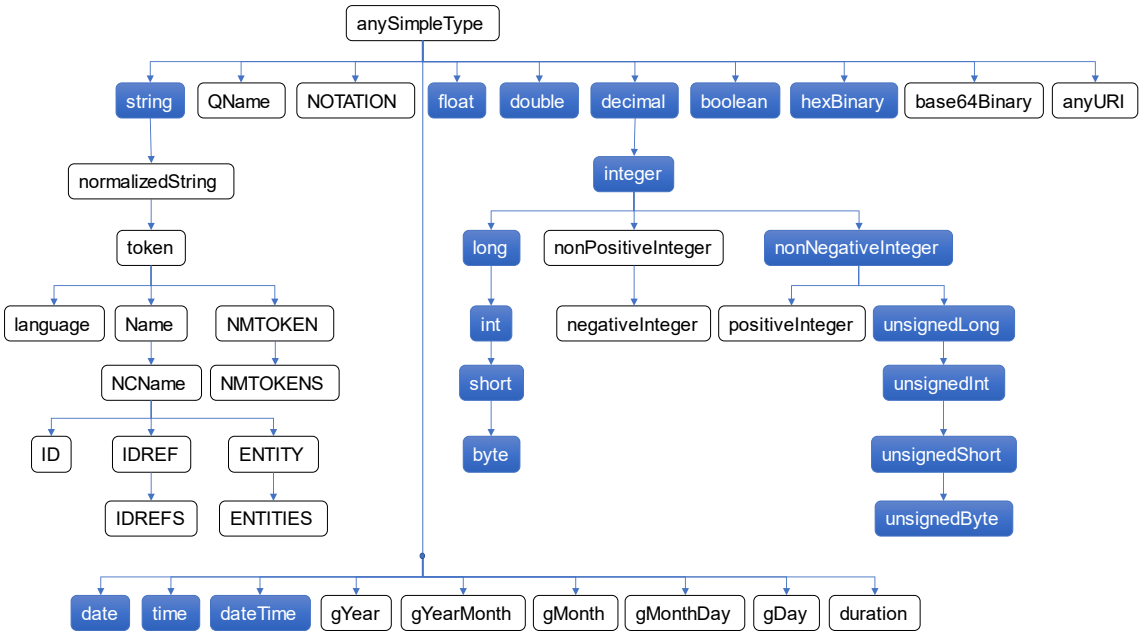

Figure 3 DFDL simple types as a subset of XML Schema types

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

- 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.)

The simple types are placed into logical type groupings as shown in this table:

Logical Type Group	Types
Number	xs:double, xs:float, xs:decimal, xs:integer, xs:nonNegativeInteger, xs:long, xs:int, xs:short, xs:byte, xs:unsignedLong, xs:unsignedInt, xs:unsignedShort, and xs:unsignedByte
String	xs:string
Calendar	xs:dateTime, xs:date, xs:time
Opaque	xs:hexBinary
Boolean	xs:boolean

Table 1: Logical type groupings

Note that DFDL does not have specific types corresponding to time intervals, nor are there special numeric types for geo-coordinates, currency, or complex numbers. These concepts must be described in DFDL using the available types.

5.2 DFDL Subset of XML Schema

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

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

DFDL Schemas consist of:

- Standard XSD namespace management
- Standard XSD import and management for multiple file schemas
- Local element declarations with dimensionality via XSD maxOccurs and XSD minOccurs.
- Global element declarations
- Complex type 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

© ISO/IEC 2024 – All rights reserved

Documents
Document Sharing Hub
Full document details: BS ISO/IEC 23415:2024

- 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 XSD minOccurs and XSD maxOccurs or with both XSD minOccurs="1" and XSD maxOccurs="1")
- 'choice' model groups (without XSD minOccurs and XSD maxOccurs or with both XSD minOccurs="1" and XSD maxOccurs="1")
- 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 XSD maxOccurs and XSD minOccurs.
- Group references without dimensionality
- Nillable attribute is "true" (that is, nillable="true" in the element declaration.)
- 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
- The xs:schema “elementFormDefault” attribute
- The xs:element “form” attribute

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
- XSD maxOccurs and XSD minOccurs on model groups (except if both are '1')
- XSD 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)

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

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

- the facets
- minOccurs, maxOccurs
- default
- fixed

[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.

[7] By reserved it is intended that conforming DFDL v1.0 implementations MUST NOT assign semantics to them.

© ISO/IEC 2024 – All rights reserved

Documents
Document Sharing Hub
Full document details: BS ISO/IEC 23415:2024

The facets and the types they are applicable to are:

- minLength maxLength (for types xs:string, and xs:hexBinary)
- pattern
- enumeration (all types except xs:boolean)
- maxInclusive, maxExclusive, minExclusive, minInclusive (for Number and Calendar types in Section 5.1)
- totalDigits (for type xs:decimal and all supported integer types descending from xs:decimal in Section 5.1)
- fractionDigits (for type xs:decimal)

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

The following sections describe these in more detail.

5.3.1 MinOccurs, MaxOccurs

XSD minOccurs and XSD maxOccurs are used in these definitions:

- An element declaration or reference where XSD minOccurs is greater than zero is said to be a required element.
- An element declaration or reference where XSD minOccurs is equal to zero is said to be an optional element.
- A required element or optional element where XSD maxOccurs is greater than 1 is also said to be an array element.

When validating, XSD minOccurs and XSD maxOccurs are used to determine the minimum and maximum valid number of occurrences of an element.

The XSD minOccurs and XSD maxOccurs values are interpreted in conjunction with the DFDL dfdl:occursCountKind property. See Section 16, Properties for Array Elements and Optional Elements, for more details.

5.3.2 MinLength, MaxLength

These facets are used:

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

5.3.3 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.

5.3.4 Pattern

- Allowed only on elements of type xs:string or types derived from it in Section 5.1.
- 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.

5.3.5 Enumeration Values

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

- Used for validation only

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

5.3.6 Default

The XSD default property is used both when parsing and unparsing, to provide the default value of an element when the situation warrants it. See 9.4 Element Defaults.

Note that the XSD fixed and XSD default properties are mutually exclusive on an element declaration.

© ISO/IEC 2024 – All rights reserved

Full document details: BS ISO/IEC 23415:2024

5.3.7 Fixed

The XSD fixed property is used in the same ways as the XSD default property but in addition:

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

Note that the XSD fixed and XSD default properties are mutually exclusive on an element declaration.

5.4 Compatibility with Other Annotation Language Schemas

A DFDL Schema only applies DFDL annotations on a subset of the XML Schema constructs. Hence,one normally thinks that a DFDL schema cannot contain any of the constructs outside of the DFDL subset. For example, the DFDL subset of XML Schema does not use attributes, hence, a DFDL schema normally would not contain attribute declarations.

There is an exception to this, however. One reason to xs:include/xs:import another XML schema document is purely for its use in validating annotations within the schema itself. Such an XML schema is describing not data, but a schema language extension of non-DFDL xs:annotation elements to be used in the rest of the schema.

Hence, the complete set of files making up a schema by way of xs:include/xs:import may include a mixture of DFDL schemas that use only the DFDL subset of XSD, as well as other XML Schemas that describe just annotations. These annotation schemas are unrestricted by the DFDL subset of XML Schema. For example, they may include elements containing xs:attribute declarations.

A DFDL processor needs a way to tell these schema files apart so that it can enforce the DFDL subset in schema files that are describing data formats and ignore the XML schema files that are for unknown annotation languages that are to be ignored by the DFDL processor.

Hence, this rule: a DFDL implementation MUST ignore any schema file included or imported by a DFDL schema if the top level xs:schema element of that included/imported schema does not have an XML namespace binding for the DFDL namespace.

© ISO/IEC 2024 – All rights reserved

Documents
Document Sharing Hub
Full document details: BS ISO/IEC 23415:2024

6 DFDL Syntax Basics

Using DFDL, a data format is described by placing special annotations at various positions within an XML schema. 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/[8]. All symbols in this namespace are reserved. DFDL implementations MUST 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 in no namespace are ignored by a DFDL processor.

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* as a whole includes all additional schema documents referenced through import and include. Generally, in this specification, when referring to the DFDL Schema this is intended to mean the schema as a whole. When referring to a specific document, the term DFDL Schema document is used.

6.2 The DFDL Annotation Elements

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

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

The *resolved set of annotations* for an annotation point is a combined set of annotations taken from:

1. a simple type definition and the base simple type it references.
2. an element declaration and the type definition from (1) it references.
3. an element reference and the global element declaration from (2) it references.
4. a group reference and the global group definition it references

Annotation Type	Annotation Element	Description
Format Annotation	dfdl:choice	Defines the physical data format properties of an xs:choice group. See Section 7.1.
	dfdl:element	Defines the physical data format properties of an xs:element and xs:element reference. See Section 7.1.
	dfdl: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.
	dfdl:group	Defines the physical data format properties of an xs:group reference. See Section 7.1.
	dfdl:property	Used in the syntax of format annotations. See Section 7.1.1.2.

[8] Note that the trailing slash is required.

© ISO/IEC 2024 – All rights reserved

Documents
Document Sharing Hub
Full document details: BS ISO/IEC 23415:2024

	dfdl:sequence	Defines the physical data format properties of an xs:sequence group. See Section 7.1.
	dfdl:simpleType	Defines the physical data format properties of an xs:simpleType. See Section 7.1.
	dfdl:escapeScheme	Defines the scheme by which quotation marks and escape characters can be specified. This is for use with delimited text formats. See Section 7.4.
Statement Annotation	dfdl:assert	Defines a test to be used to ensure the data are well formed. Assert is used only when parsing data. See Section 7.5
	dfdl:discriminator	Defines a test to be used when resolving choice branches and optional element occurrences. A dfdl:discriminator is used only when parsing data. See Section 7.6
	dfdl:newVariableInstance	Creates a new instance of a variable. See Section 7.7.2
	dfdl:setVariable	Sets the value of a variable whose declaration is in scope See Section 7.7.3
Defining Annotation	dfdl:defineEscapeScheme	Defines a named, reusable escapeScheme See Section 7.3
	dfdl: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
	dfdl: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 2 - DFDL Annotation Elements

DFDL defining annotation elements may only appear at *top-level*, that is, as annotation children of the xs:schema element. The order of their appearance does not matter, nor does their position relative to other children of the xs:schema.

6.3 DFDL Properties

A DFDL *property* is a specific DFDL construct that tells the DFDL processor some characteristic about the data format.

Properties carried on the component format annotations (See Section 7.1) are called *format properties*. A format property that is used to describe a physical characteristic of a component is called a *representation* property.

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

- Enumeration
  The property value is an XSD xs:token the value of which is one of the allowed values listed in the property description.

  Example: the dfdl:lengthKind property, which has values taken from “delimited”, “fixed”, “explicit”, “implicit”, “prefixed”, “pattern”, and “endOfParent”. For example:

  ```
  lengthKind='delimited'
  ```

- DFDL string literal (Section 6.3.1):
  The property value represents a sequence of literal bytes or characters which represent data which appears in the data stream. The value type is a restriction of the XSD xs:token that further disallows the space character. DFDL entities must be used to express whitespace in a DFDL String Literal.

  Example: the dfdl:terminator property, which expresses characters or bytes to be found in the data stream to mark the termination of an element or model group instance. An example terminator might be:

  ```
  terminator='%NL;'
  ```

  This uses DFDL’s string-literal character class entity syntax (see Section 6.3.1.3) to express that the element or model group is terminated by a line ending in the data stream.

© ISO/IEC 2024 – All rights reserved

Documents
Document Sharing Hub
Full document details: BS ISO/IEC 23415:2024

- DFDL expression (Section 6.3.2)
  The property is an xs:string the value of which is a DFDL expression that returns a value derived from other property values and/or from the DFDL Infoset. Leading and trailing whitespace is trimmed for DFDL expressions.

  Example: the dfdl:occursCount property takes an expression which commonly looks in the Infoset via an expression, to obtain the count from another element. An example dfdl:occursCount property might be:

  ```
  occursCount='{ ../hdr/count }'
  ```

- DFDL regular expression (Section 6.3.3)
  The property is an xs:string the value of which 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. Note that leading and trailing whitespace is not trimmed and is part of the regular expression value.

  Example: the dfdl:lengthPattern property takes a regular expression which is used to scan the data stream for matching data. An example might be:

  ```
  lengthPattern="\w{1,5};"
  ```

  This scans the data stream for from 1 to 5 word-characters followed by a semi-colon character.

- 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.

  Example: the dfdl:nilValue property can be used to provide a logical value that if it matches the element's logical value is used to indicate the data is nilled. For example for an element of type xs:int:

  ```
  nilValue='0'
  ```

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

  Example: The dfdl:escapeSchemeRef property refers to a named escape scheme definition via its qualified name. For example:

  ```
  escapeSchemeRef='ex:backslashScheme'
  ```

Some properties accept a list or union of types

- List of DFDL String Literals or Logical Values
  The property value is a whitespace 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.

  Example: The dfdl:separator property below indicates that the items of a sequence are separated either by a comma or a tab character.

  ```
  separator=', %HT;'
  ```

- 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.

  Example: Below are two examples of the dfdl:length property. One uses an expression that resolves to an unsigned integer, the other a literal unsigned integer.

  ```
  length='{ xs:unsignedInt(../hdr/len) }'

  length='14'
  ```

- Union of types.
  The property value is a union of two or more types. The type is often dependent on the value of another property.

© ISO/IEC 2024 – All rights reserved

Documents
Document Sharing Hub

Full document details: BS ISO/IEC 23415:2024

For example, dfdl:nilValue can be a List of DFDL String Literals or a List of Logical Values depending on dfdl:nilKind. Another example is the dfdl:alignment property which can have as its value an unsigned integer or the distinguished enum value 'implicit'.

6.3.1 DFDL String Literals

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 character set 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
- 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.

Details on how a string literal is matched against the data stream for parsing are given in Appendix C: Processing of DFDL String literals.

Empty String: The special DFDL entity %ES; is provided for describing an empty string or an empty byte sequence. The %ES; entity is the only way to do this. A DFDL string literal with value "" (the empty string) is usually invalid. There are a few properties that explicitly allow an empty DFDL String Literal, and these properties assign a property-specific meaning to the empty string value.

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,a DFDL schema may describe 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 the schema indicates that an item is "," (comma) separated and specifies 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 dfdl:encoding representation property. Hence, in this case the processor 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.

6.3.1.2 DFDL Character Entities, Character Class Entities, and Byte Values 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, DFDL character entities can express common non-printable characters or code points, such as 0x00, that are not

© ISO/IEC 2024 – All rights reserved

Documents
Document Sharing Hub
Full document details: BS ISO/IEC 23415:2024

valid in XML documents. DFDL entities are based on XML entities, which can also be used in a DFDL schema. Examples:

```
separator='%HT;'
terminator='%WSP*;//'
fillByte='%#x00;'
textStringPadCharacter='%#x7F;'
```

In some cases, regular XML character entities may be used instead. For example, the above '%#x7F;' could be expressed as '&#x7F;' but this is not always the case. There is no way in XSD to express the character code 0 (i.e., the ASCII NUL code point), even as an XML character entity; hence, one must often use DFDL character entities like '%#x00;' above, or their named equivalents. The DFDL string literal syntax allows the author to always use DFDL character entity syntax instead of jumping back and forth between XSD character entities and DFDL character entities.

The following grammar gives the syntax of DFDL String Literals generally, including the various kinds of entities.

```
DfdlStringLiteral       ::=  (DfdlStringLiteralPart)+ | DfdlESEntity

DfdlStringLiteralPart   ::=  LiteralString | DfdlCharEntity | DfdlCharClass |
                             ByteValue

LiteralString           ::=  A string of literal characters

DfdlCharEntity          ::=  DfdlEntity | DecimalCodePoint |
                             HexadecimalCodePoint

DfdlCharClass           ::=  '%' DfdlCharClassName ';'

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

DfdlEntity              ::=  '%' DfdlEntityName ';'

DecimalCodePoint        ::=  '%#' [0-9]+ ';'

HexadecimalCodePoint    ::=  '%#x' [0-9a-fA-F]+ ';'

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'

DfdlCharClassName       ::=  DfdlNLEntity | DfdlWSPEntity | DfdlWSPStarEntity |
                             DfdlWSPPlusEntity

DfdlNLEntity            ::=  'NL'

DfdlWSPEntity           ::=  'WSP'

DfdlWSPStarEntity       ::=  'WSP*'

DfdlWSPPlusEntity       ::=  'WSP+'

DfdlESEntity            ::=  'ES'
```

Table 3 DFDL Character Entity, Character Class Entity, and Byte Value 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.

© ISO/IEC 2024 – All rights reserved

Documents
Document Sharing Hub
Full document details: BS ISO/IEC 23415:2024

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 4 DFDL Entities

© ISO/IEC 2024 – All rights reserved

Documents
Document Sharing Hub
Full document details: BS ISO/IEC 23415:2024

6.3.1.3 DFDL Character Class Entities in DFDL String Literals

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

Mnemonic	Meaning	Unicode Character Code(s)
NL	Newline On parse any one of the single 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+000A LF U+000D CR U+000D U+000A CRLF U+0085 NEL U+2028 LS
WSP	Single whitespace On parse any whitespace 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 a Processing Error if no whitespace character is found. On unparse a space (U+0020) is output.	Same as WSP
ES	Empty String Used in whitespace separated lists when empty string is one of the values.	

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. The syntax is in Table 3 DFDL Character Entity, Character Class Entity, and Byte Value Entity Syntax above. Example:

```
%#rFF;
```

In this notation the "r" can be thought of as short for "raw", as byte value entities are said to denote "raw bytes".

© ISO/IEC 2024 – All rights reserved

Documents
Document Sharing Hub

Full document details: BS ISO/IEC 23415:2024

6.3.2 DFDL Expressions

Some DFDL properties allow DFDL expressions (see Section 18 DFDL 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

- Discard any leading and trailing whitespace.
- 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.

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

6.3.3 DFDL Regular Expressions

Some properties expect a regular expression to be specified. The DFDL Regular Expression language is defined in Section 19, 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.

© ISO/IEC 2024 – All rights reserved

Documents
Document Sharing Hub
Full document details: BS ISO/IEC 23415:2024

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 2 - 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 6 DFDL Component Format Annotations

Below are a few examples followed by sections which describe each kind of annotation element in detail. Here is an example of DFDL component format annotation, specifically use of dfdl:element on an xs:element declaration:

```
<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>
```

Note that in the above, the DFDL annotation lives inside this surrounding context of xs:annotation and xs:appinfo elements. This is just the standard XSD way of doing annotations. The source attribute is an identifier that separates different families of appinfo annotations.

Belowa dfdl:format annotation is used inside a dfdl:defineFormat annotation to define a named reusable set of format properties that can be referenced from another format annotation.

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

      <dfdl:defineFormat name="baseFormat">
        <dfdl:format byteOrder="bigEndian" encoding="ascii"/>
      </dfdl:defineFormat>

    </xs:appinfo>
  </xs:annotation>
  ...
```

© ISO/IEC 2024 – All rights reserved

Documents
Document Sharing Hub
Full document details: BS ISO/IEC 23415:2024

```
</xs:schema>
```

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 8.1.2 Providing Defaults for DFDL properties.)

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

        <dfdl:format
           representation="binary"
           byteOrder="bigEndian"
           encoding="ascii"/>

    </xs:appinfo>
  </xs:annotation>
  ...
</xs:schema>
```

7.1.1 Property Binding Syntax

A *property binding* is the syntax in a DFDL schema that gives a value to a property. Up to this point, the examples in this document have all used a specific syntax for property bindings called *attribute form*. However, the format properties may be specified in any one of three forms:

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

A DFDL property may be specified using any of the forms with the following exceptions:

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

It is a Schema Definition Error if the same property is specified in more than one form. That is, there is no priority ordering where one form takes precedent over another.

7.1.1.1 Property Binding Syntax: Attribute Form

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

```
 PropertyName="Value"
```

For example:

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

This is the attribute form of property binding.

7.1.1.2 Property Binding Syntax: Element Form

The representation properties can sometimes have complex syntax, so an element form for individual property bindings is provided to ease syntactic expression difficulties. The annotation element is 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>
```

© ISO/IEC 2024 – All rights reserved

Documents
Document Sharing Hub
Full document details: BS ISO/IEC 23415:2024

```
    </xs:appinfo>
  </xs:annotation>
```

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

Here is an example where a delimiter has a syntax that overlaps with what XML comments look like. Use of XML's CDATA bracketing makes this less clumsy to express than using XML escape characters:

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

7.1.1.3 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; attribute form must be used instead. 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.

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>
```

The above show use of short-form property binding syntax for annotating elements and sequences.

7.1.2 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 a valid value for certain properties.

For example, in non-delimited text data formats, 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

© ISO/IEC 2024 – All rights reserved

Documents
Document Sharing Hub
Full document details: BS ISO/IEC 23415:2024

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

To avoid error-prone redundant expression of properties in DFDL schemas, a collection of DFDL properties can be given a name so that they are reusable by way of a *format reference*.

One or more dfdl:defineFormat annotation elements can appear within the annotation children of the xs:schema element.

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 becomes 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="baseFormat" >
        <dfdl:format representation="text"
                     encoding="ascii" />
      </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 Using/Referencing a Named Format Definition: The dfdl:ref Property

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

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

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

7.2.2 Inheritance for dfdl:defineFormat

A dfdl:defineFormat declaration can inherit from another named format definition by use of the dfdl: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.

An example format that inherits from a named format definition is:

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

Conceptually, the dfdl: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 the discussion assumes inheritance is fully flattened. That is, all dfdl:ref inheritance is first removed by flattening before any other examination of properties occurs.

It is a Schema Definition Error if use of the dfdl:ref property results in a circular path.

© ISO/IEC 2024 – All rights reserved

Documents
Document Sharing Hub
Full document details: BS ISO/IEC 23415:2024

7.3 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 becomes 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:escapeScheme 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 a dfdl:escapeScheme for reuse from other places. It does not cause any use of the representation properties it contains to describe any actual data.

7.3.1 Using/Referencing a Named escapeScheme Definition

A named, reusable, escape scheme is used by referring to its name from a dfdl: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>
```

7.4 The dfdl:escapeScheme Annotation Element

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 properties 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 pad characters are trimmed and on unparsing before pad characters are added.

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

The properties of dfdl:escapeScheme are defined in Section 13.2.1 The dfdl:escapeScheme Properties.

© ISO/IEC 2024 – All rights reserved

Documents
Document Sharing Hub
Full document details: BS ISO/IEC 23415:2024

7.5 The dfdl:assert Statement Annotation Element

The dfdl:assert statement annotation element is used to assert truths about a DFDL model that are used when parsing to ensure that the data are well-formed. They are not used when unparsing.

There is a critical distinction between dfdl:assert checks and XSD validation checks.

The dfdl:assert checks guide parsing and the creation of the DFDL Infoset by causing Processing Errors on failure. Conversely XSD validation inspects the values within the Infoset. Validation failures never affect whether the parser is able to produce a DFDL Infoset.

The dfdl:assert checks are performed even when validation is off.

Examples of dfdl:assert elements are below:

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

7.5.1 Properties for dfdl:assert

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 either a Processing Error or a Recoverable Error to be issued, as specified by the failureType property of the dfdl:assert.

The testKind property 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 test expression or test pattern 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 these schema components:

- 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.

If one of the resolved set of asserts for a schema component is unsuccessful, and the failureType of the assert is ‘processingError’, then no further asserts in the set are executed.

Property Name	Description
testKind	Enum (optional) Valid values are 'expression', 'pattern' Default value is 'expression' Specifies whether a DFDL expression or DFDL regular expression pattern is used in the dfdl:assert.

© ISO/IEC 2024 – All rights reserved

Documents
Document Sharing Hub
Full document details: BS ISO/IEC 23415:2024

	Annotation: dfdl:assert
test	DFDL Expression Applies when 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 must be a descendent of this element. 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 testKind is 'expression' or not specified, and an expression is not supplied by either the value of the dfdl:assert element or the value of the test attribute. Annotation: dfdl:assert
testPattern	DFDL Regular Expression Applies when 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 pattern matching of the regular expression reads data that cannot be decoded into characters of the current encoding, then the behavior is controlled by the dfdl:encodingErrorPolicy property. See Section 11.2.1 Property dfdl:encodingErrorPolicy for details. 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 testKind is 'pattern', and a pattern is not supplied by either the value of the dfdl:assert element or the value of the testPattern property. 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 or DFDL Expression Defines text to be used as a diagnostic code or for use in an error message, when the assert is unsuccessful. The DFDL Expression must return type xs:string. Any element referred to by the message expression must have already been processed or must be a descendent of this element. There is special treatment for errors that occur while evaluating the message expression. See below for details. 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

© ISO/IEC 2024 – All rights reserved

Documents
Document Sharing Hub
Full document details: BS ISO/IEC 23415:2024

Table 7 dfdl:assert properties

Example of a dfdl:assert with a message expression:

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

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

If a Processing Error or Schema Definition Error occurs while evaluating the message expression, a Recoverable Error is issued to record this error (containing implementation-dependent content), then processing of the assert continues as if there were no problem and in a manner consistent with the failureType property, but using an implementation-dependent substitute message.

7.6 The dfdl:discriminator Statement Annotation Element

DFDL discriminator statement annotations are used during parsing to:

1. resolve *points of uncertainty* (choices, optional elements, array repetition) that cannot be resolved by speculative parsing. See Section 9.1 Parser Overview.
2. remove ambiguity during speculative parsing
3. improve diagnostic behavior when a DFDL parser encounters malformed data.

Discriminators are not used during unparsing.

A DFDL discriminator may contain a test 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. A discriminator may alternatively contain a test regular expression pattern and the discriminator is successful if the test pattern matches with non-zero length and is unsuccessful (or fails) if there is no match or a zero-length match.

A discriminator determines the existence or non-existence of a schema component in the data stream. If the discriminator is successful, then the component is said to be *known to exist*, and any subsequent errors do not cause backtracking at the nearest point of uncertainty. Details of the behavior of a DFDL parser and the role of discriminators are given in Section 9.3 Parsing Algorithm.

Discriminators can also be used to force a resolution earlier during the parsing of a model group so that subsequent parsing errors are treated as Processing Errors of a known schema component rather than a failure to find that schema component. This may greatly improve the efficiency of DFDL parsing in some implementations, as well as improving the diagnostic information provided by a DFDL parser when given malformed data.

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.

7.6.1 Properties for dfdl:discriminator

Within a dfdl:discriminator, the testKind property 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 the test expression or test pattern 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 these schema components:

© ISO/IEC 2024 – All rights reserved

Documents
Document Sharing Hub
Full document details: BS ISO/IEC 23415:2024

- 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:assert annotations, 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 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 its descendants have been processed or when a Processing Error occurs when processing this element or its descendants. It is a Schema Definition Error if testKind is 'expression' or not specified, and an expression is not supplied by either the value of the dfdl:discriminator element or the value of the test attribute. Annotation: dfdl:discriminator
testPattern	DFDL Regular Expression Applies when 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 pattern matching of the regular expression reads data that cannot be decoded into characters of the current encoding, then the behavior is controlled by the dfdl:encodingErrorPolicy property. See Section 11.2.1 Property dfdl:encodingErrorPolicy for details. 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 testKind is 'pattern', and a pattern is not supplied by either the value of the dfdl:discriminator element or the value of the testPattern property. 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'

© ISO/IEC 2024 – All rights reserved

Documents
Document Sharing Hub
Full document details: BS ISO/IEC 23415:2024

	Annotation: dfdl:discriminator
message	String or DFDL Expression Defines text to be used as a diagnostic code or for use in an error message, when the discriminator is unsuccessful. The DFDL Expression must return type xs:string. Any element referred to by the message expression must have already been processed or must be a descendent of this element. There is special treatment for errors that occur while evaluating the message expression. See below for details. Annotation: dfdl:discriminator

Table 8 dfdl:discriminator properties

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

If a Processing Error or Schema Definition Error occurs while evaluating the message expression, a Recoverable Error is issued to record this error (containing implementation-dependent content), then processing of the discriminator continues as if there were no problem, but in the case of failure using an implementation-dependent substitute message.

Examples of dfdl:discriminator annotations:

```
<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>
```

© ISO/IEC 2024 – All rights reserved

Documents
Document Sharing Hub
Full document details: BS ISO/IEC 23415:2024

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

7.7 DFDL Variable Annotations

DFDL Variables provide a means for communication and parameterization within a DFDL schema. Use of variables increases the modularity of a schema by enabling some parts of a schema to be parameterized so that they are reusable.

There are 3 DFDL annotation elements associated with DFDL variables:

- dfdl:defineVariable - defines a variable and creates a global instance of it.
- dfdl:newVariableInstance - creates a scoped instance of a variable.
- dfdl:setVariable - assigns the value of a variable instance, which can be global or scoped.

Variables are defined at the top-level of a schema and have a specific simple type.

A distinction is made between the variable as defined, and an *instance* of the variable where a value can be stored.

The dfdl:defineVariable annotation defines the name, type, and optionally default value for the variable. It is like defining a class of variables, instances of which actually store values. The dfdl:defineVariable also introduces a single unique global instance of the variable. Additional instances may be allocated in a stack-like fashion using dfdl:newVariableInstance which causes new instances to come into existence upon entry to the scope of a model group, and these instances go away on exit from the same.

DFDL variables only *vary* in the sense that different instances of the same variable can have different values. A single instance of a variable only ever takes on a single value. Each variable instance is a *single-assignment* location for a value[9]. Once a variable instance's value has been read, it can never be assigned again. If it has not yet been assigned, and its default value has not been read, then a variable instance can be assigned *once* using dfdl:setVariable.

Variables are used by referencing them in DFDL expressions by prefixing their QNames with '$'.

More information about variables and how they work operationally is in Section 18.2 Variables. The remaining sub-sections of this section focus only on the variable-related DFDL annotations and their syntax.

7.7.1 dfdl:defineVariable Annotation Element

A global 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). If the defaultValue is given by an expression that expression must not contain any relative path (otherwise it is a Schema Definition Error).

Note that the syntax supports both a defaultValue attribute and the default value being specified by the element value. Only one or the other may be present (otherwise it is a Schema Definition Error). To set the default value to "" (empty string), the defaultValue attribute syntax must be used, or the expression { "" } must be used as the element value.

Note also that the value of the name attribute is an NCName (non-colon name - that is, may not have a prefix). 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.

[9] The rationale for single-assignment variables is to keep DFDL schemas *declarative* by preventing variables from being used as algorithmic accumulators. See the Appendix B: Rationale for Single-Assignment Variables.

© ISO/IEC 2024 – All rights reserved

Full document details: BS ISO/IEC 23415:2024

A default *instance* of the variable is automatically created (with global scope) at the start of a DFDL parse or unparse. Additional instances of a variable can be created with the scope of other schema components. See Section 7.7.2 The dfdl:newVariableInstance Statement Annotation Element.

The external property 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 is used as the global default value overriding any defaultValue specified on the dfdl:defineVariable annotation. The mechanism by which the processor provides this value is *implementation-defined*.

A variable instance gets its value either from the default value provided in the dfdl:defineVariable definition, from an external binding of the variable if the definition has the external attribute, from a dfdl:setVariable statement (See Section 7.7.3, The dfdl:setVariable Statement Annotation Element), or from the default value of a dfdl:newVariableInstance statement (See Section 7.7.2 The dfdl:newVariableInstance Statement Annotation Element.)

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 MUST be evaluated before processing of the data stream begins.

A defaultValue expression can refer to other variables but not to the Infoset (so no path locations).When a defaultValue expression references other variables, the referenced variables each must either have a defaultValue or be external. It is a Schema Definition Error otherwise.

If a defaultValue expression references another variable then the single-assignment nature of variables prevents the referenced variable's value from ever changing, that is, it is considered to be a read of the variable's value, and once read, a variable's value cannot be changed.

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.

7.7.1.1 Examples

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

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

7.7.1.2 Predefined Variables

The following variables are predefined, and their names are in the DFDL namespace (http://www.ogf.org/dfdl/dfdl-1.0/)

Name	Type	Default value	External
dfdl:encoding	xs:string	'UTF-8'	true
dfdl:byteOrder	xs:string	'bigEndian'	true
dfdl:binaryFloatRep	xs:string	'ieee'	true
dfdl:outputNewLine	xs:string	'%LF;'	true

Table 9 Pre-defined variables

These variables are expected to be commonly set externally so are predefined for convenience. Below the DFDL encoding property is being set to the value of a DFDL expression (between "{" and "}"), and that expression just returns the value of the dfdl:encoding variable which is being referenced as $dfdl:encoding below.

```
      <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>
```

7.7.2 The dfdl:newVariableInstance Statement Annotation Element

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

© ISO/IEC 2024 – All rights reserved

Documents
Document Sharing Hub
Full document details: BS ISO/IEC 23415:2024

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

All instances share the same name, type, and default value if provided, but they have distinct storage for separate values using a stack-like mechanism where a new instance is introduced for a model group. These new instances are associated with a schema component using dfdl:newVariableInstance. These instances have the lifetime of the schema component. While that schema component is being parsed/unparsed, the new variable instance is used and other scoped variable instances for the same variable are not available.

Since an initial global instance is created when the variable is defined, the use of dfdl:newVariableInstance is optional.

The dfdl:newVariableInstance annotation can be used on a group reference, sequence or choice only. It is a Schema Definition Error otherwise.

The lifetime 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 property 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 inherits 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 is created when the instance is created. The value overrides 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 default value being specified by the annotation element value. Only one or the other may be present. (Schema definition error otherwise.)

To set the default value to "" (empty string), the defaultValue attribute syntax must be used, or the expression { "" } must be used as the element value.

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. The order of execution is specified in Section 9.5 Evaluation Order for Statement Annotations.

There is no short form syntax for creating variable instances.

7.7.2.1 Examples

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

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

7.7.3 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 simple type, 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 property is a QName. That is, it may be qualified with a namespace prefix.

© ISO/IEC 2024 – All rights reserved

Documents
Document Sharing Hub
Full document details: BS ISO/IEC 23415:2024

The syntax supports both a value attribute and the 'value' being specified by the element value. Only one or the other may be present (otherwise it is a Schema Definition Error). To set the value to "" (empty string), the value attribute syntax must be used, or the expression { "" } must be used as the element value.

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 allows 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 expression must not contain forward references to elements which have not yet been processed.

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 branches of a choice or as a result of speculative parsing. In this case the variable state is also rewound. See Section 9 DFDL Processing Introduction.

A dfdl:setVariable overrides 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 (different name and/or namespace) and it is a Schema Definition Error otherwise. The order of execution is specified in Section 9.5 Evaluation Order for Statement Annotations.

There is no short form syntax for variable assignment.

7.7.3.1 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="delimiter"> {.} </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. The variable delimiter (in the default namespace) is also being assigned the same value using other syntax.

© ISO/IEC 2024 – All rights reserved

Documents
Document Sharing Hub
Full document details: BS ISO/IEC 23415:2024

8 Property Scoping and DFDL Schema Checking

8.1 Property Scoping

8.1.1 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 10 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 10 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).

8.1.2 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 the encoding property. The value `'ASCII'` is the default value for the `title` element, but then it is overridden by the locally defined `utf-8` value for the encoding property, which takes precedence.

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

© ISO/IEC 2024 – All rights reserved

Documents
Document Sharing Hub
Full document details: BS ISO/IEC 23415:2024

```
  <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>
```

8.1.3 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.

8.1.4 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 (a) an empty working set of "explicit" properties, and (b) 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).

© ISO/IEC 2024 – All rights reserved

Documents
Document Sharing Hub
Full document details: BS ISO/IEC 23415:2024