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)

COPYRIGHT PROTECTED DOCUMENT

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

Contents

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.

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.

```
  </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.

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>
```

```
        </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.

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.