DFDL Introduction For Beginners – Lesson 1

This document, DFDL Schema Tutorial, provides an easily approachable description of the Data Format Definition Language 1.0 (DFDL for short), and should be used alongside the formal descriptions of the language contained in the DFDL 1.0 Specification. The intended audience of this document includes application developers whose programs read and write DFDL-described data, and DFDL schema authors who need to know about the features of the language. The text assumes that you have a basic understanding of XML 1.0, Namespaces in XML and XML Schema. Each section of the tutorial introduces new features of the language, and describes those features in the context of concrete examples.

Introduction

Lesson1. Explains what DFDL is, why DFDL is useful, and when to use DFDL. The first DFDL examples are introduced.

Lesson 2. Introduces the fundamentals of the DFDL language. The subset of XML Schema components used by DFDL is discussed as well as the special DFDL annotations that are used with each component.

Lesson 3. Here we learn more about DFDL annotations, the pros and cons of short form versus atribute form, as well as annotation placement and scoping.

Lesson 4. Modeling text data. Examples showing the basic annotations needed for character based data.

Lesson 5. Modeling numeric data. Examples showing the basic annotations needed for numeric elements. These examples show how DFDL can be used for numbers in a textual context, for binary numbers in scientific formats, and for formats used in COBOL copybooks.

Lesson 6. Modeling date and time data. Examples showing the basic annotations needed for dates, times and timestamps.

Lesson 7. Padding and trimming of data.

Lesson 8. Want to default in values for missing data? Need to model out-of-band data values? What does empty string represent? This lesson walks through examples using nil and default settings.

Lesson 9. Explains how to handle delimiters that occur in data values by using both escape characters and escape blocks. Steps through setting up an escape scheme.

Lesson 10. Choices – life is more than sequences! Learn how to set up alternative branches in your model. Discusses DFDL specific requirements for choices as well as a preview of discriminators.

Lesson 11. Have components in your data where the order of the data is not fixed? This lesson takes you through an example of a floating component within a group. Then we will make the entire group unordered.

Lesson 12. DFDL expressions. Several examples will show this powerful feature of the language, from a simple path expression to a complex conditional expression.

Lesson 13. Creating dynamic models. Building on top of expressions, learn how to use DFDL variables to add in complex control of your DFDL properties and other settings.

Lesson 14. Making assertions about your data. Discover the uses of asserts and discriminators, and the difference between them. Concepts such as rule placement, and timing of triggering will be discussed via example.

Lesson 15. Elements with calculated values and Hidden Elements

Lesson 16. Bits versus bytes – don't worry if your data is expressed in terms of bits rather than bytes – DFDL can handle that as well.

The tutorial is a non-normative document, meaning that it does not provide a definitive specification of the DFDL language. The examples and other explanatory material in this document are provided to help you understand DFDL but they may not always provide definitive answers. In such cases, you will need to refer to the DFDL 1.0 Specification, and to help you do this, many links pointing to the relevant parts of the specification are provided.

Conventions used:

New terms will be introduced in *italics*.
Examples will be presented in blue shading.

WHAT IS DFDL?

Data Format Definition Language or DFDL is pronounced like the flower daffodil: ˈdæfədɪl. It is a language designed to describe the format of data. Specifically, it is designed to describe the format of data in a way that is independent of the format itself. The idea is that you choose an appropriate data representation for an application based on its needs and then describe the format using DFDL so that multiple programs can directly interchange the described data. That is, DFDL is *not* a format for data; it is a way of describing *any* data format.

DFDL is intended for data commonly found in scientific and numeric computations, as well as record-oriented representations found in commercial data processing. DFDL can be used to describe legacy data files, to simplify transfer of data across domains without requiring global standard formats, or to allow third-party tools to easily access multiple formats.

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

Note that DFDL is specifically _not_ intended to be used to describe XML, which already has well-defined ways to describe it. However, the DFDL language is built upon many of the principals of XML and is designed to make XML tooling available for use with non-XML data.

Schema – A model, a framework, a plan. We will be using the generic term schema to mean a model of some data.

XML Schema or XSD – A model used specifically to describe the structure and content of XML data/instance documents.

Instance Document – A term that describes the entire stream of data that we are processing in this 'run' or 'instance'.

DFDL Schema – A model used specifically to describe the structure and content of non-XML data/instance document.

A DFDL schema uses a subset of XML schema constructs, so a DFDL schema is actually a well-formed and valid XML. However, a DFDL schema does not describe data that is XML, it describes data that is **not** XML.

How can a DFDL schema (which itself is an XML schema) describe non-XML data? The answer is 'annotations'.

Annotations – A way to provide additional information in an XML schema. They are ignored by XML parsers. They can be used to add documentation or to add additional information that will be used by application programs.

Well-formed XML – XML instance document which conforms to all the Well-formedness constrains in the W3C XML Recommendation. In other words, it follows the basic rules of XML.

Valid XML – Not only does this XML instance document follow the rules of well-formedness, it also conforms to the rules imposed by its XML schema.

WHEN TO USE DFDL

DFDL should be used to describe non-XML data. XML data is different from non-XML data in a variety of ways. One obvious way is that with XML you always know that your data will start with an XML start tag and end with an XML end tag. So an XML schema doesn't need to tell you how to find the beginning and end of each piece of data.

Example Tags – Showing XML start and end tags:
<song>**One More Than Two Visually Challenged Rodents**</song>

Description:
***<song>** is the start tag*
***</song>** is the end tag*

The data that is 'song' is:
One More Than Two Visually Challenged Rodents

A DFDL schema describes non-XML data which might not have start tags and end tags so needs other ways to define in the schema where data starts and ends. There are many types of structures for data, for example, sometimes data is made up of fixed length components, and sometimes data has delimiters to tell us where pieces of the data begin and end.

Fixed Length – The length of a portion of data is known at the time of data modeling (design time) and is always the same. For example, in the US a State Code is always 2 characters in length.

Delimiters (also called Markup or Syntax) - Clues in the data that tell us where pieces of the data begin and end. For example in 'written English' a period or full stop tells us that we are at the end of a sentence.

FIRST SET OF DFDL EXAMPLES:

The best way to understand how DFDL works is to look at a couple of small examples. We will model the same logical data in different ways.

1. A sequence of fixed-length data elements
2. A sequence of delimited data elements

Description of data for all address examples:
element 1: houseNumber
element 2: street
element 3: city
element 4: state

Example Address 1 – address with fixed length data components:
0000000118   Ridgewood Circle   WashingtonNY

Description:
In the above example we have an instance document which is fixed length of 62 composed of :
10 character houseNumber
25 character street
25 character city
2 character state

Example Address 2 – address with delimited data components:
0000000118*Ridgewood Circle*Washington*NY

Description:
In the above example we have an instance document which has variable sized components, infix delimited by an asterisk (*):
piece 1: houseNumber
piece 2: street
piece 3: city
piece 4: state

Variable length – The length of a portion of data is not known at the time the data is being modeled (design time) but is determined when data is being processed (run-time).

Infix Delimited – Data in which a delimiter or separator appears in between each component. This delimiter is not found before the first component, not is it found after the last component.

Looking at the address examples we can see exactly the same data content[1] expressed in different ways. With DFDL we are operating under the principal that we already know what our data looks like and that we want to model it in the easiest way possible – we want to describe the format that we already have. If our data is non-XML and we want to use an open-standard modeling language, then we will want to use DFDL.

Now that we've seen different ways to express our address data we can compare different ways to model it.

Modeling – the process of specifying the structure and content of your data file. In the above example we say that the XML schema is the model for the XML instance document. The process of creating a model or a schema is sometimes called modeling.

DFDL MODELING FIXED LENGTH DATA

To show one way to model fixed length data we will start by modeling Example Address 1 –address with fixed length data components.

We need to describe where each piece of data begins and ends. We will do this by using DFDL.

Looking again at our fixed length data example.

Example Address 1 continued– address with fixed length data components:

```
0000000118        Ridgewood Circle             WashingtonNY
```

Description:
In the above example we have an instance document which is composed of:

10 character houseNumber
25 character street
25 character city

2 character state

Note: When looking at the data we don't see anything in the data that names the 'root' of the data. In our case we will call the entire piece of data an 'address', so out root is 'address'.
We also do not see anything in the data which shows us where each piece of data begins and end – we need to know that our first 10 characters of data is the houseNumber in order to understand this data.

There are multiple ways to represent this structure in DFDL. In the following example we are using the 'DFDL Long Form Annotations'. There are portions of the DFDL schema that are not shown in the below example in order to make the concepts clearer and simpler to start (but I promise, there will be much more detail in future lessons!). However, a link to the full schema is available in the "Lesson 1" zip file. In this version we are using annotations to add in the lengths of our fixed length elements:

```
<xs:element name="address">
  <xs:annotation>
    <xs:appinfo source="http://www.ogf.org/dfdl/">
      <dfdl:element lengthKind="implicit"/>
    </xs:appinfo>
  </xs:annotation>
  <xs:complexType>
    <xs:sequence>
      <xs:element name="houseNumber" type="xs:string">
        <xs:annotation>
          <xs:appinfo source="http://www.ogf.org/dfdl/">
            <dfdl:element lengthKind="explicit" length="10"/>
          </xs:appinfo>
        </xs:annotation>
      </xs:element>
      <xs:element name="street" type="xs:string">
        <xs:annotation>
          <xs:appinfo source="http://www.ogf.org/dfdl/">
            <dfdl:element lengthKind="explicit" length="25"/>
          </xs:appinfo>
        </xs:annotation>
      </xs:element>
      <xs:element name="city" type="xs:string">
        <xs:annotation>
          <xs:appinfo source="http://www.ogf.org/dfdl/">
            <dfdl:element lengthKind="explicit" length="25"/>
          </xs:appinfo>
        </xs:annotation>
      </xs:element>
      <xs:element name="state" type="xs:string">
        <xs:annotation>
          <xs:appinfo source="http://www.ogf.org/dfdl/">
            <dfdl:element lengthKind="explicit" length="2"/>
          </xs:appinfo>
        </xs:annotation>
      </xs:element>
    </xs:sequence>
  </xs:complexType>
```

```
</xs:element>
```

This same structure can also be represented in DFDL using ‘short form’ attributes. Notice how much more compact the short form is than the long form! In later lessons we will describe why you would select to use short form, as well as the advantages of long form. For now you just need to realize that DFDL provides this flexibility.

```
1 <xs:element name="address" dfdl:lengthKind="implicit">
2   <xs:complexType>
3     <xs:sequence dfdl:sequenceKind="ordered">
4       <xs:element name="houseNumber" type="xs:string"
              dfdl:lengthKind="explicit" dfdl:length="10"/>
5       <xs:element name="street" type="xs:string"
              dfdl:lengthKind="explicit"  dfdl:length="25"/>
6       <xs:element name="city" type="xs:string"
              dfdl:lengthKind="explicit"  dfdl:length="25"/>
7       <xs:element name="state" type="xs:string"
              dfdl:lengthKind="explicit"  dfdl:length="2"/>
8     </xs:sequence>
9   </xs:complexType>
10</xs:element>
```

There are a few new concepts shown in the above example that we will highlight.

If you noticed that the dfdl models in this example look a great deal like XML schema then you would be correct. Even though our data is not XML, DFDL uses XML schema to model it

- Line 1 establishes an element named address with one DFDL property on it setting the DFDL lengthKind to “implicit”, this means that the length of element ‘address’ is determined by the length of its children.
- Line 3 is establishing an unnamed sequence within element address. It also holds a DFDL property of sequenceKind equals “ordered”. This property describes the sequence components (to follow) as always being in the order in which they are listed in the model. ‘houseNumber’ is always first, followed by ‘street’, ‘city’ and ending with ‘state’.
- Line 4 lists the first element of the sequence, houseNumber. On this element there is one DFDL property specifying that its length is 10. There is another DFDL property, lengthKind, which is set to“explicit”. This means that the length of this element is determined explicitly using the DFDL length property.
- Line 5 defines the second element in the sequence as element street. The DFDL properties on this element describes it as having an explicit length of 25.
- Line 6 defines the third element in the sequence as element city. The DFDL properties on this element describes it as having an explicit length of 25.

- Line 7 defines the fourth element in the sequence as element state. The DFDL properties on this element describes it as having an explicit length of 2.

The data content from our example may be represented in an infoSet format such as:

```
Infoset:
    Address:
        houseNumber:0000000118
        street:Ridgewood Circle
        city:Washington
        state:NY
```

Notice that the spaces have been trimmed from the values in the above infoset, this is due to trimming properties set in the model. We can not see that the space is set up as the pad/trim character in the section of the model that we are displaying. However, in later lessons we will learn how to set up global properties and get full details on padding and trimming.

Infoset – Also known as 'Information Set' is an abstract data model that describes the information content of a document.

DFDL MODELING VARIABLE LENGTH DELIMITED DATA

We started by modeling Example Address 1 – address with fixed length data components. Now we want to model the same data but our components are not fixed length. In this case we can find the beginning and end of each part of the data by looking at markup in the instance document. Here is our example data again.

Example Address 2 – address with delimited data components:
0000000118*Ridgewood Circle*Washington*NY

There is an asterisk (*) between each piece of data in this instance document. As there is no asterisk before the first piece of data (houseNumber) or after the last piece of data (state) we say that the data is infix delimited by '*'.

When we model this structure in DFDL we will not be specifying element lengths in our model but will be specifying the delimiter and its location. Such a DFDL schema may look like (short form):

```
1  <xs:element name="address" dfdl:lengthKind="implicit">
2    <xs:complexType>
3      <xs:sequence dfdl:sequenceKind="ordered"
                    dfdl:separator="*"
                    dfdl:separatorPosition="infix"
                    dfdl:separatorPolicy="required">
```

```
4        <xs:element name="houseNumber" type="xs:string"
                dfdl:lengthKind="delimited"/>
5        <xs:element name="street" type="xs:string"
                dfdl:lengthKind="delimited"/>
6        <xs:element name="city" type="xs:string"
                dfdl:lengthKind="delimited"/>
7        <xs:element name="state" type="xs:string"
                dfdl:lengthKind="delimited"/>
8      </xs:sequence>
9    </xs:complexType>
10 </xs:element>
```

- Line 1 establishes an element named address with one DFDL property on it setting the DFDL lengthKind to "implicit", meaning that the length of address is determined by the length of its children.
- Line 3 is establishing an unnamed sequence within element address. It also holds a DFDL property of sequenceKind equals "ordered". This property describes the sequence components (to follow) as always being in the order in which they are listed in the schema. houseNumber is always first, followed by street, city and ending with state. This line also holds three properties necessary to define the delimiter used in the data. A delimiter between the components of a sequence is called a separator is DFDL. Accordingly, the DFDL separator property is set to the asterisk "*", the DFDL separatorPosition property is "infix" and the DFDL separatorPolicy property is "required" meaning that the "*" is always expected, even when the data is missing.
- Line 4 lists the first element of the sequence, houseNumber. On this element there is one DFDL property, lengthKind, which is set to"delimited". This means that the length of this element is determined by looking at delimiters in the data.
- Line 5 defines the second element in the sequence as element street. The DFDL property on this element describes it as delimited.
- Line 6 defines the third element in the sequence as element city. The DFDL property on this element describes it as delimited.
- Line 7 defines the fourth element in the sequence as element state. The DFDL property on this element describes it as delimited.

Once parsed by a DFDL parser the data content may be represented in an infoSet format such as:

```
Infoset:
    Address:
        houseNumber:0000000118
        street:Ridgewood Circle
        city:Washington
        state:NY
```

Notice that the Example 1 infoSet and the Example 2 infoSet look the same. This is because in both example instance documents the data is logically the same – it is just presented differently (with fixed sizes in example 1 versus with delimiters in example 2). Once we strip away the delimiters and the padding we are left with the same data. This distinction between physical data and logical data will be very important (and useful) as we progress though our understanding of DFDL.

Now that we've seen a very small set of what DFDL can do, we need to start exploring how to exploit it fully. Our next lesson will introduce fundamentals of the DFDL language and the concept of DFDL annotations.

Footnotes:

[1] we will not discuss extra spaces in the data at this time. See lesson 7 'Padding and Trimming'