DFDL Introduction For Beginners

Lesson 1: Introduction

Version	Author	Date	Change
1	S Fetzer	2011-01-24	Created
2	S Hanson	2011-03-30	Improved
3	S Hanson	2011-03-30	Corrected
4	S Hanson	2012-09-21	Reflect lessons in existence
5	S Hanson	2013-09-03	Match latest spec

This document, DFDL Schema Tutorial, provides an easily approachable description of the Data Format Description 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.

The following lessons are available:

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. Describes the syntax and rules for the DFDL properties that are carried on DFDL annotations and used to model data.

Lesson 4. Shows how to model basic fixed length data, variable length data, and data with initiators (tags) and terminators, using DFDL properties.

Lesson 5. Shows how to model data that contains alternatives, known in DFDL as a ‘choice’.

Lesson 6. Shows how to model data that may optionally occur in the data stream, or that repeats multiple times in the data stream.

The following lessons are planned:

Lesson 7. Covers the modeling of all kinds of text data, including strings, numbers, calendars and Booleans.

Lesson 8. Covers the modeling of all kinds of binary data, including strings, numbers, calendars and Booleans.

Lesson 9. Shows how padding and trimming are handled in DFDL, especially useful when dealing with fixed length data.

Lesson 10. 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 11. Explains how to handle delimiters that occur in data values by using either escape characters and escape blocks. Steps through setting up an escape scheme.

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.

Lesson 15. Bits versus bytes – don't worry if your data is expressed in terms of bits rather than bytes – DFDL can handle that as well.

Lesson 16. Have components in your data where the order of the data is not fixed? This lesson shows how to model unordered data.

Lesson 17. Elements with calculated values and hidden elements.

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 *italic font*.

Examples will be presented in boxed `courier font`.

What is Data Format Description Language?

Data Format Description Language or DFDL is pronounced like the flower 'daffodil'. 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 schema document.

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.

When to use DFDL

DFDL should be used to describe general text or binary 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 XML showing 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.

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). Often used in conjunction with delimiters.

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 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 variable-length delimited data elements

Description of logical 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:

```
000118Ridgewood Circle    Rochester           NY
```

In the above example we have an instance which is fixed length of 48 composed of :

6 digit houseNumber
20 character street
20 character city
2 character state

Example Address 2 – address with delimited data components:

```
118*Ridgewood Circle*Rochester*NY
```

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

Infix Delimited – Data in which a delimiter or separator appears in between each component. This delimiter is not found before the first component, nor is it found after the last component.

Looking at the address examples we can see exactly the same logical data content[1] expressed in different ways. With DFDL we are operating under the principle 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 can 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.*

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 – address with fixed length data components:

```
000118Ridgewood Circle    Rochester            NY
```

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 ends – we need to know that our first 6 characters of data is the 'houseNumber' in order to understand this data.

There are a few ways to lay out this structure in DFDL. In the following example we are using what are known as 'short form annotations' which is the most compact layout. There are portions of the DFDL schema that are not shown in the example below in order to make the concepts clearer and simpler to start (there will be much more detail in future lessons). In this version we are using annotations to add in the lengths of our fixed length elements:

```
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="6"/>
5       <xs:element name="street" type="xs:string"
              dfdl:lengthKind="explicit"  dfdl:length="20"/>
6       <xs:element name="city" type="xs:string"
              dfdl:lengthKind="explicit"  dfdl:length="20"/>
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 length to be "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 saying that the sequence is "ordered". So '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 are two DFDL properties specifying that it has an explicit length of 6.
- Line 5 defines the second element in the sequence as 'street'. The DFDL properties on this element describes it as having an explicit length of 20.
- Line 6 defines the third element in the sequence as 'city'. The DFDL properties on this element describes it as having an explicit length of 20.
- 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     '118'
        street          'Ridgewood Circle'
        city            'Rochester'
        state           'NY'
```

Notice that leading zeros and trailing spaces have been trimmed from the values in the infoset. It is possible to do this using other DFDL properties (not shown), as we will see in later lessons.

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

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 when 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 data, known as 'delimiters' in DFDL. Here is our example data again.

Example Address 2 – address with delimited data components:

```
118*Ridgewood Circle*Rochester*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 instead will be specifying the delimiter and its location. Such a DFDL schema, again in 'short form' may look like:

```
1  <xs:element name="address" dfdl:lengthKind="implicit">
2    <xs:complexType>
3      <xs:sequence dfdl:sequenceKind="ordered"
                    dfdl:separator="*"
                    dfdl:separatorPosition="infix">
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 length to be "implicit", meaning 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 property saying that the sequence is "ordered". So 'houseNumber' is always first, followed by 'street', 'city' and ending with 'state'. This line also holds three other DFDL properties necessary to define the delimiter used in the data. A delimiter between the components of a sequence is called an 'infix separator' in DFDL. Accordingly, the separator property is set to the asterisk "*", its position is set to "infix".
- Line 4 lists the first element of the sequence, 'houseNumber'. On this element there is one DFDL property saying its length is "delimited", meaning

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.

The data content from our example may be represented in an ‘infoset’ format such as:

Infoset:

```
address
    houseNumber     ‘118’
    street          ‘Ridgewood Circle’
    city            ‘Rochester’
    state           ‘NY’
```

Notice that the infosets in Example 1 and Example 2 look the same. This is because in both example instance documents the data is logically the same – it is just presented differently physically (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.