DFDL Introduction for Beginners

Lesson 4: Describing the Structure of the Data

In lessons 1 to 3 we have learnt about the purpose and basic syntax of DFDL. In this lesson we will learn how to describe the basic structure of the data. We learn the most common ways to identify the elements and in later lessons we will learn how to describe the physical format of the individual elements.

In lesson 1 we saw some examples of fixed length and delimited data structures. We will build on those examples to introduce more details.

Modeling fixed length data

Example 1: Address with fixed length data elements

Data stream

```
000118Ridgewood Circle    Rochester            NY
```

DFDL schema

```
1 <xs:schema … xmlns:dfdl="http://www.ogf.org/dfdl/dfdl-1.0/">
2  <xs:annotation>
3    <xs:appinfo source="http://www.ogf.org/dfdl/" >
4      <dfdl:format representation="text"
                    lengthKind="explicit"
                    lengthUnits="bytes"
                    encoding="ASCII" />
5    </xs:appinfo>
6  </xs:annotation>

7  <xs:element name="address" dfdl:lengthKind="implicit">
8   <xs:complexType>
9     <xs:sequence dfdl:sequenceKind="ordered">
10     <xs:element name="houseNumber" type="xs:string"
              dfdl:length="6" />
11     <xs:element name="street" type="xs:string"
              dfdl:length="20" />
12     <xs:element name="city" type="xs:string"
              dfdl:length="20" />
13     <xs:element name="state" type="xs:string"
              dfdl:length="2" />
14     </xs:sequence>
15   </xs:complexType>
16  </xs:element>
17 </xs:schema>
```

Infoset

```
address
    houseNumber(string)   '000118'
    street(string)        'Ridgewood Circle    '
    city(string)          'Rochester           '
    state(string)         'NY'
```

The simplest way to model fixed length data is to use a `dfdl:lengthKind` of "explicit", `dfdl:length` to indicate the length and `dfdl:lengthUnits` to indicate the units.

Since all the simple elements in the data are fixed length, in line 4 we have set `dfdl:lengthKind` to a default of "explicit" to save it having to be specified on each element. The actual length is specified by the `dfdl:length` property but as this varies for each element it is specified on each separately on lines 10 to 13. The `dfdl:lengthUnits` property specifies the units that the `dfdl:length` property represents and can be either "bytes", "characters" or "bits". When "characters" is specified then the element is a fixed number of characters in the encoding given by the `dfdl:encoding` property. For single byte fixed length encodings then "bytes" and "characters" lengths are the same. Lengths in bits are described in a later lesson.

Notice that leading zeros and trailing spaces have not been trimmed from the data. DFDL provides properties to control trimming and padding, which are described in a later lesson.

On line 7 element address has `dfdl:lengthKind` "implicit" defined. For a complex element `dfdl:lengthKind` "implicit" means that the length of the element is the combined length of its children and so avoids the user having manually add up all the child lengths.

A `dfdl:lengthKind` "implicit" can also be used to provide an implied length for some simple elements depending on the physical representation; more details are provided in a later lesson.

Modeling fixed length delimited data

Sometimes, even though an element is fixed length, it is part of a sequence that uses separators.

Example 2: Address with fixed length, delimited data elements

Data stream

```
000118*Ridgewood Circle    *Rochester           *NY
```

DFDL schema

```
1 <xs:schema … xmlns:dfdl="http://www.ogf.org/dfdl/dfdl-1.0/">

2  <xs:annotation>
3    <xs:appinfo source="http://www.ogf.org/dfdl/" >
4      <dfdl:format representation="text"
                    lengthKind="explicit"
                    lengthUnits="bytes"
                    encoding="ASCII" />
5    </xs:appinfo>
6  </xs:annotation>

7  <xs:element name="address" dfdl:lengthKind="implicit">
8    <xs:complexType>
9      <xs:sequence dfdl:sequenceKind="ordered"
                    dfdl:separator="*"
                    dfdl:separatorPosition="infix">

10     <xs:element name="houseNumber" type="xs:string"
              dfdl:length="6" />
11     <xs:element name="street" type="xs:string"
              dfdl:length="20" />
12     <xs:element name="city" type="xs:string"
              dfdl:length="20" />
13     <xs:element name="state" type="xs:string"
              dfdl:length="2" />

14     </xs:sequence>
15   </xs:complexType>
16 </xs:element>

17 </xs:schema>
```

Infoset

```
    address
        houseNumber(string)   '000118'
        street(string)        'Ridgewood Circle    '
        city(string)          'Rochester           '
        state(string)         'NY'
```

On line 9, the `xs:sequence` has a separator "*" defined which is used to separate each of the children of the sequence. The DFDL separator properties are explained in the next example. It is worth noting that when an element is fixed length and has delimiters (that is, when `dfdl:lengthKind` is not "delimited") the length is used to extract the element content from the data stream rather than scanning for a delimiter.

Modeling variable length delimited data

Example 3: Address with variable length delimited data elements:

Data stream

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

DFDL schema

```
1 <xs:schema … xmlns:dfdl="http://www.ogf.org/dfdl/dfdl-1.0/">

2  <xs:annotation>
3    <xs:appinfo source="http://www.ogf.org/dfdl/" >
4      <dfdl:format representation="text"
                    lengthKind="delimited"
                    encoding="ASCII" />
5    </xs:appinfo>
6  </xs:annotation>

7  <xs:element name="address" dfdl:lengthKind="implicit">
8    <xs:complexType>
9      <xs:sequence dfdl:sequenceKind="ordered"
                    dfdl:separator="*"
                    dfdl:separatorPosition="infix" >
10       <xs:element name="houseNumber" type="xs:string"
/>
11       <xs:element name="street" type="xs:string" />
12       <xs:element name="city" type="xs:string" />
13       <xs:element name="state" type="xs:string" />
14     </xs:sequence>
15   </xs:complexType>
16 </xs:element>

17 </xs:schema>
```

Infoset

```
    address
        houseNumber(string)   '118'
        street(string)        'Ridgewood Circle'
        city(string)          'Rochester'
        state(string)         'NY'
```

Variable length data is usually modeled using `dfdl:lengthKind` "delimited" which means that an element is terminated by:

- the element's terminator (if specified)
- an enclosing element or sequence's separator or terminator
- the end of an enclosing element of known length
- the end of the data stream

Here, the `dfdl:length` property is not specified, and the data stream is scanned for all possible terminating delimiters in order to extract the element content.

Most commonly, the element will be terminated by the separator of the enclosing `xs:sequence`; however the last element in the sequence may be terminated by an enclosing element's delimiter or the end of the data stream. Some formats will require a separator between each element in the sequence which is modeled by `dfdl:separatorPosition` "infix". Other formats may require a additional separator to be before the first element or after the last element which is modeled by `dfdl:separatorPosition` "prefix" and "postfix", respectively.

Modeling data with initiators and terminators

Example 4: Address with Initiated and terminated data elements

Data stream

```
[house:118*street:Ridgewood Circle*city:Rochester*state:N
Y]
```

DFDL schema

```
1 <xs:schema … xmlns:dfdl="http://www.ogf.org/dfdl/dfdl-1.0/">

2  <xs:annotation>
3    <xs:appinfo source="http://www.ogf.org/dfdl/" >
4      <dfdl:format representation="text"
                    lengthKind="delimited"
                    encoding="ASCII" />
5    </xs:appinfo>
6  </xs:annotation>

7  <xs:element name="address" dfdl:lengthKind="implicit"
         dfdl:initiator="[" dfdl:terminator="]">
8    <xs:complexType>
9      <xs:sequence dfdl:sequenceKind="ordered"
                    dfdl:separator="*"
                    dfdl:separatorPosition="infix" >
10       <xs:element name="houseNumber" type="xs:string"
                   dfdl:initiator="house:" />
11       <xs:element name="street" type="xs:string"
                   dfdl:initiator="street:" />
12       <xs:element name="city" type="xs:string"
                   dfdl:initiator="city:" />
13       <xs:element name="state" type="xs:string"
                   dfdl:initiator="state:" />
14     </xs:sequence>
15   </xs:complexType>
16 </xs:element>

17 </xs:schema>
```

Infoset

```
    address
        houseNumber(string)    '118'
        street(string)         'Ridgewood Circle'
        city(string)           'Rochester'
        state(string)          'NY'
```

Initiators (sometimes called ‘tags’) are often used to aid identification of an element, particularly in text based EDI standards. Terminators are used to designate the end of an element, typically when there is no separator. Initiators and terminators can be specified for simple or complex, fixed length or delimited, elements.

In the example, the overall address data now starts with a “[“ and ends with a “]”, and each element in the address is prefixed by its name followed by a colon. Accordingly, on line 7 the `address` element is complex and has a `dfdl:initiator` "[" and a `dfdl:terminator` "]" defined. On line 11 the simple element `street` has a `dfdl:initiator` "street:" defined. Though not shown here, initiators and terminators can also be defined on sequences and choices such as the `xs:sequence` on line 9.

If defined then the initiator and terminator must be present in the data stream except when the element contains a special value such as the empty string or nil as explained in a later lesson. The terminator, if defined, is mandatory but may be omitted from the very last element in a data stream if property `dfdl:documentFinalTerminatorCanBeMissing` is "yes" .

DFDL separators and terminators are similar in that they both indicate the end of an element and in many formats either could be used to model a terminating delimiter. In general a terminator should be used when the delimiter is considered a fundamental part of the element and needs to be present wherever the element declaration is reused in the schema. They should also be used when the delimiter for each element in a sequence is different. Separators should be used when the delimiter depends on the context of where the element is used. For example the same element declaration may be reused in multiple sequences that use different separators.

Summary

In this lesson we have looked at the most common ways you can define the structure of the data. We have seen how to define structures that contain fixed length and variable length elements. We have seen how initiators can be used to identify which elements are in the data stream which is useful, for example, when there is an unknown number of a repeating element which is described in a later lesson.

In later lessons we will look at other types of data structures including dynamic structures that determine the length of elements using other elements within the data stream itself or where the length immediately precedes the element.