DFDL Introduction For Beginners

Lesson 3: DFDL properties

Version	Author	Date	Change
1	S Hanson	2011-01-24	Created
2	S Hanson	2011-03-30	Updated
3	S Hanson	2012-09-21	Corrections for errata
4	S Hanson	2013-09-03	Match latest spec

In lesson 2 we learned that in the DFDL language, XML Schema conveys the logical structure of the data format being modeled, and DFDL properties carried on DFDL annotations describe the physical aspects of that data. In this lesson we take a closer look at DFDL properties and explain the syntax and rules that govern them.

DFDL properties

In order to define the physical *representation* of the data, we will add DFDL properties to the model we are constructing to describe our data. A comprehensive list of DFDL properties is available, allowing everything about the physical data to be described. For this lesson we will concentrate on the rules governing properties, and not the individual properties themselves which are described in later lessons. The vast majority of DFDL properties describe a physical characteristic of an object, and are known as *representation properties*. For now it is sufficient to note that representation properties can be divided into two broad categories:

Framing – how values are extracted from the data stream. Includes properties for alignment, length, and delimiters.

Content – how values are represented in the data stream. Includes properties for data values and padding.

DFDL property types

A DFDL property has one of the following types:

- *DFDL string literal*
  The property value is a string that represents a sequence of literal bytes or characters which appear in the data stream.

  ```
  dfdl:separator="*"
  ```

  A special syntax called *DFDL entities* is provided to handle non-printable characters and raw hexadecimal bytes.

  ```
  dfdl:separator="%CR;%LF;"
  ```

- *Enumeration*
  The property value is one of a set of allowed strings, listed in the property description.

```
dfdl:lengthKind="delimited"
```

- *Logical Value*
  The property value is a string that represents a logical value. The type of the logical value is one of the XML Schema simple types. For example, a non-negative integer.

```
dfdl:alignment="4"
```

- *DFDL expression*
  The property value is an expression that returns a value. Expressions are covered in detail in lesson 12, but in brief they are XPath 2.0 compliant, are able to refer to other elements in the data, and must be enclosed in curly braces { }.

```
dfdl:occursCount="{../itemCount}"
```

- *QName*
  The property value is an XML Qualified Name, that is, a name in a namespace. For example, a reference to an object in the XML Schema.

```
dfdl:prefixLengthType="tns:twoBytePrefixType"
```

- *Regular expression*
  The property value is a regular expression that can be applied to the literal characters which appear in the data stream. Only used by a couple of properties.

```
dfdl:lengthPattern="[0-9]{5}"
```

Sometimes a single value of a single type is not enough to model the behavior we want from a property. DFDL allows some properties to have several values, and allows some properties to have more than one type.

List properties
DFDL allows some properties to be a space-separated list of values. For example, a structure in the data might permit either of two different characters to be its separator. To represent this, DFDL allows the dfdl:separator property to be a list. In this example, either a * or a % in the data is accepted as the separator.

```
dfdl:separator="* %"
```

Union properties
DFDL allows some properties to be a choice of two types, that is, the value can be of one type or the other. For example, elements in the data can be

aligned on an explicit boundary, or can be implicitly aligned according to their data type. To represent this, DFDL allows the property type to be a union. In this example, the dfdl:alignment property may either be a logical value (non-negative integer) or the enumeration 'implicit'.

```
dfdl:alignment="4"

dfdl:alignment="implicit"
```

Where one of the types in the union is an expression, the expression must return a value that matches the other type in the union.

DFDL property syntax

DFDL properties may be expressed in one of three equivalent syntaxes – *attribute* form, *element* form and *short* form. We will use the variable length Address example from lessons 1 and 2 to illustrate this.

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

Attribute form
In lesson 2 we showed what the 'houseNumber' element from the 'address' example might look like once it has a DFDL annotation added, if 'houseNumber' is a variable length right-justified ASCII text integer.

```
<xs:element name="houseNumber" type="xs:int"/>
  <xs:annotation>
    <xs:appinfo source=”http://www.ogf.org/dfdl/” >
      <dfdl:element representation=”text”
                    textNumberRep=”standard”
                    lengthKind=”delimited”
                    encoding=”ASCII”
                    textNumberPattern=”##0”
    </xs:appinfo>
  </xs:annotation>
</xs:element>
```

When DFDL properties are carried as attributes on a DFDL annotation element in this way, the DFDL properties are said to be in attribute form (sometimes called *long* form).

Element form
In this form, the value of a DFDL property is the value of a DFDL property annotation element.

```
<xs:element name="houseNumber" type="xs:int"/>
  <xs:annotation>
    <xs:appinfo source=”http://www.ogf.org/dfdl/” >
      <dfdl:element>
         <dfdl:property name=”representation”>text</dfdl:property>
         <dfdl:property name=”textNumberRep”>standard</dfdl:property>
         <dfdl:property name=”lengthKind”>delimited</dfdl:property>
         <dfdl:property name=”encoding”>ASCII</dfdl:property>
```

```
        <dfdl:property name="textNumberPattern">##0</dfdl:property>
    </xs:appinfo>
  </xs:annotation>
</xs:element>
```

Element form is useful when the DFDL property value contains characters that are not allowed in XML attributes. You can even use the CDATA tag if the value is malformed from an XML point of view.

Short form
In this form, the DFDL properties are attributes in the http://www.ogf.org/dfdl/dfdl-1.0/ namespace and are carried directly on the XML Schema objects themselves. This is a more concise coding syntax and you will see it used a great deal from now on!

```
<xs:element name="houseNumber" type="xs:int"
            dfdl:representation="text"
            dfdl:textNumberRep="standard"
            dfdl:lengthKind="delimited"
            dfdl:encoding="ASCII"
            dfdl:textNumberPattern="##0" />
```

You can mix all property forms on an object, as long as a specific property only appears once.

Providing defaults for DFDL properties

When you start to create a DFDL schema model of your data, you will soon notice that many of your data elements are very similar, and that consequently the elements, groups and simple types need to carry a common set of DFDL property values. For example, in the variable length 'address' example, all elements have text representation, the same encoding, and are delimited.

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

  <xs:element name="address" dfdl:lengthKind="implicit">
    <xs:complexType>
      <xs:sequence dfdl:sequenceKind="ordered"
                   dfdl:encoding="ASCII"
                   dfdl:separator="*"
                   dfdl:separatorPosition="infix"
                   dfdl:separatorSuppressionPolicy="never">
        <xs:element name="houseNumber" type="xs:int"
                    dfdl:representation="text"
                    dfdl:textNumberRep="standard"
                    dfdl:lengthKind="delimited"
                    dfdl:encoding="ASCII"
                    dfdl:textNumberPattern="##0" />
        <xs:element name="street" type="xs:string"
                    dfdl:lengthKind="delimited"
                    dfdl:encoding="ASCII" />
        <xs:element name="city" type="xs:string"
                    dfdl:lengthKind="delimited"
                    dfdl:encoding="ASCII" />
```

```
        <xs:element name="state" type="xs:string"
                    dfdl:lengthKind=”delimited”
                    dfdl:encoding=”ASCII” />
      </xs:sequence>
    </xs:complexType>
  </xs:element>

</xs:schema>
```

The format annotation

DFDL provides an annotation called *format* which allows you to set up values that act like defaults. A single format annotation can be created at the top level of the XML Schema (which means it belongs to the schema object itself). Any properties carried by the format annotation in this manner provide defaults for *all* the objects in the schema.

Here's the 'address' example again, but with some common properties moved into such a format annotation.

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

  <xs:annotation>
    <xs:appinfo source=”http://www.ogf.org/dfdl/” >
      <dfdl:format representation=”text” lengthKind="delimited"
                   encoding=”ASCII” />
    </xs:appinfo>
  </xs:annotation>

  <xs:element name="address" dfdl:lengthKind="implicit">
    <xs:complexType>
      <xs:sequence dfdl:sequenceKind="ordered"
                   dfdl:separator="*"
                   dfdl:separatorPosition="infix"
                   dfdl:separatorSuppressionPolicy="never">
        <xs:element name="houseNumber" type="xs:int"
                    dfdl:textNumberRep=”standard”
                    dfdl:textNumberPattern=”##0” />
        <xs:element name="street" type="xs:string" />
        <xs:element name="city" type="xs:string" />
        <xs:element name="state" type="xs:string" />
      </xs:sequence>
    </xs:complexType>
  </xs:element>

</xs:schema>
```

In the example, the dfdl:representation, dfdl:encoding, and dfdl:lengthKind properties have effective defaults for all objects in the schema that need those properties. If an object provides a value for dfdl:representation, dfdl:encoding or dfdl:lengthKind then that overrides the default.

The DFDL properties on the format annotation may be expressed in attribute form and element form only (short form is not allowed).

RULE: Individual properties in DFDL do *not* have built-in default values. If a property is applicable to an object then a value must be provided somewhere in the schema.

RULE: The defaults from a format element do not apply to objects in other DFDL schemas accessed using xs:include or xs:import statements. Defaults apply only to the objects within the same schema.

Because DFDL properties do not have defaults, some properties need a special syntax to indicate that they are not used. The empty string "" is used for this purpose. Examples are dfdl:initiator, dfdl:terminator and dfdl:separator.

DFDL property scoping rules

RULE: A DFDL property only applies to the object on which it is declared.

In the 'address' example, the dfdl:encoding property of a sequence applies only to the sequence itself, and not to any contained elements. So the sequence's dfdl:encoding says whether each separator between child elements is ASCII or EBCDIC, etc, but says nothing about the child elements themselves. The elements for each child have their own dfdl:encoding property which says whether the element content is ASCII or EBCDIC, etc.

Creating reusable sets of DFDL properties

The defineFormat annotation
DFDL provides an annotation called *defineFormat* which allows the declaration of reusable bundles of DFDL properties. DFDL defineFormat annotations are created at the top level of the XML Schema. Any number of these bundles can be created in a DFDL schema, and they are identified by their 'name' attribute. Once created, they may be referenced from other objects in the schema, using a special dfdl:ref property. Here's the 'address' example again, but this time with the common properties moved into a defineFormat annotation named 'common'.

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

  <xs:annotation>
    <xs:appinfo source="http://www.ogf.org/dfdl/" >
      <dfdl:defineFormat name="common">
        <dfdl:format representation="text" lengthKind="delimited"
                     encoding="ASCII" />
      </dfdl:defineFormat>
    </xs:appinfo>
  </xs:annotation>

  <xs:element name="address" dfdl:lengthKind="implicit">
    <xs:complexType>
      <xs:sequence dfdl:ref="common"
                   dfdl:sequenceKind="ordered"
                   dfdl:separator="*"
```

```
                    dfdl:separatorPosition="infix"
                    dfdl:separatorSuppressionPolicy="never">
        <xs:element name="houseNumber" type="xs:int"
                    dfdl:ref=”common”
                    dfdl:textNumberRep=”standard”
                    dfdl:textNumberPattern=”##0” />
        <xs:element name="street" type="xs:string"
                    dfdl:ref=”common” />
        <xs:element name="city" type="xs:string"
                    dfdl:ref=”common” />
        <xs:element name="state" type="xs:string"
                    dfdl:ref=”common” />
      </xs:sequence>
    </xs:complexType>
  </xs:element>

</xs:schema>
```

Notice that the defineFormat annotation contains a *format* element that actually carries the properties. Syntactically this is the same as the format annotation introduced earlier.

An object in the schema uses the dfdl:ref property to ‘pull in’ the properties in a defineFormat annotation. Those properties are then combined with the properties defined locally on the object. If a property appears both locally and via `ref`, that’s ok, the local one is used in preference and the one via dfdl:ref is ignored.

Only one dfdl:ref property can ever appear on an object, but you can create chains of defineFormat annotations to assemble larger groups of properties.

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

  <xs:annotation>
    <xs:appinfo source=”http://www.ogf.org/dfdl/” >
      <dfdl:defineFormat name="base">
        <dfdl:format representation=”text” encoding=”ASCII” />
      </dfdl:defineFormat>

      <dfdl:defineFormat name="common">
        <dfdl:format ref=”base” lengthKind="delimited" />
      </dfdl:defineFormat>
    </xs:appinfo>
  </xs:annotation>
…
</xs:schema>
```

In the example above, the dfdl:representation and dfdl:encoding properties have been moved into a separate defineFormat named ‘base’. The defineFormat ‘common’ pulls in the properties of ‘base’, using the dfdl:ref property of its child format element. Any object using its own dfdl:ref property to access ‘common’ will get the properties of both ‘base’ and ‘common’.

The DFDL properties on the format annotation within a defineFormat may be expressed in attribute form and element form only (short form is not allowed).

DFDL properties on referenced objects

When modeling the physical aspects of a particular data element, you can sometimes be faced with a choice of where exactly to place your DFDL properties. Here's an example.

In lesson 2 we said that DFDL allows you to create your own simple types by deriving from the built-in types. Here's what that might look like for the 'houseNumber' element from the 'address' example, if we decided that house numbers could only be positive numbers.

```
<xs:element name="houseNumber" type="houseNumberType"/>

<xs:simpleType name="houseNumberType">
  <xs:restriction base="xs:int">
    <xs:minInclusive value="1" />
  </xs:restriction>
</xs:simpleType>
```

DFDL lets you add properties to elements and simple types, so where should you place the DFDL properties that model the 'houseNumber' element?

One approach is to place the DFDL properties on the element.

```
<xs:element name="houseNumber" type="houseNumberType"
            dfdl:representation="text"
            dfdl:textNumberRep="standard"
            dfdl:lengthKind="delimited"
            dfdl:encoding="ASCII"
            dfdl:textNumberPattern="##0" />

<xs:simpleType name="houseNumberType">
  <xs:restriction base="xs:int">
    <xs:minInclusive value="1" />
  </xs:restriction>
</xs:simpleType>
```

Alternatively, the DFDL properties could be placed on the referenced simple type.

```
<xs:element name="houseNumber" type="houseNumberType" />

<xs:simpleType name="houseNumberType"
               dfdl:representation="text"
               dfdl:textNumberRep="standard"
               dfdl:lengthKind="delimited"
               dfdl:encoding="ASCII"
               dfdl:textNumberPattern="##0" >
  <xs:restriction base="xs:int">
    <xs:minInclusive value="1" />
  </xs:restriction>
</xs:simpleType>
```

Both approaches are equally valid and model the element in the same way. Placing the properties on the simple type is often a good idea as it gives you a model with better reuse, as the type could then be used by several elements.

You can even split the DFDL properties between the element and the simple type. In this case, all the properties are combined into a single set.

There are several instances in DFDL where properties on objects and references may need to be combined in this way:

- An element and its referenced simple type restriction
- An element reference and its referenced global element
- A group reference and the sequence or choice in its referenced global group
- A simple type restriction and its base simple type restriction

RULE: When combining properties on objects and references, an individual property can only appear *once* (whether defined locally or pulled in via dfdl:ref).

For example you are not allowed to specify dfdl:encoding on both an element and the simple type it references.

Putting DFDL properties into action

In this lesson we have seen how to specify DFDL properties locally, as defaults using a format annotation, in named defineFormat annotations, and how to combine properties with those on referenced objects. The DFDL specification provides an algorithm that specifies how the effective set of DFDL properties for an object is obtained, taking all these techniques into account. It may be summarized as follows:

1. Independently for an object and a reference, merge local properties with those obtained via dfdl:ref
2. Combine the merged properties from the object and the reference
3. If any properties are missing, use defaults from format annotation, with reference's defaults taking precedence

At the end of all that, if a DFDL property is needed by the object and the property has not got a value, it is an error. Remember, there are no built-in defaults!

Design pattern for DFDL schema

Using the rules learned in this lesson, it is possible to create a design pattern for DFDL schemas that minimizes the declaration of DFDL properties, resulting in a compact and uncluttered model. Here's the 'address' example updated to illustrate this.

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

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

      <!– Declare common property values -->
      <dfdl:defineFormat name="base">
        <dfdl:format representation="text" encoding="ASCII"
                     initiator="" terminator="" separator=""
                     lengthKind="delimited"
                     sequenceKind="ordered"
                     separatorPosition="infix"
                     separatorSuppressionPolicy="never" … />
        </dfdl:format>
      </dfdl:defineFormat>

      <!– Set the common property values as defaults -->
      <dfdl:format ref=”base” />

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

  <xs:element name="address" dfdl:lengthKind="implicit">
    <xs:complexType>
      <xs:sequence dfdl:separator="*" >
        <xs:element name="houseNumber" type="houseNumberType" />
        <xs:element name="street" type="xs:string" />
        <xs:element name="city" type="xs:string" />
        <xs:element name="state" type="xs:string" />
      </xs:sequence>
    </xs:complexType>
  </xs:element>

  <xs:simpleType name="houseNumberType"
                 dfdl:textNumberRep=”standard”
                 dfdl:textNumberPattern=”##0” >
    <xs:restriction base=”xs:int”>
      <xs:minInclusive value=”1” />
    </xs:restriction>
  </xs:simpleType>

</xs:schema>
```

Notice how the commonly used property values declared in the named defineFormat annotation are referenced from the format annotation via dfdl:ref. This has the effect of turning all those common property values into defaults for all the objects in the schema. The only local property values needed are those that differ from the defaults.

A further refinement is to move the named defineFormat annotation into a separate dedicated DFDL schema. This may then be included or imported into other DFDL schema, maximizing its reuse potential.