DFDL Introduction For Beginners

Lesson 2: DFDL Language Basics

We have seen in lesson 1 how DFDL is not an entirely new language. Its foundation is XML Schema 1.0. Although XML Schema was created as a way of specifying XML documents, it turns out to be very good at describing the logical structure of all kinds of data. DFDL therefore uses XML Schema constructs for describing the logical structure of non-XML data, and uses XML Schema's annotation facility to describe the physical structure of the same data.

DFDL and XML Schema

XML Schema is a powerful language that is designed for modeling XML. When you look at the constructs in XML Schema, it turns out that many of them are quite specific to XML. DFDL has chosen to use a subset of the constructs in XML Schema that is just enough to model non-XML data, in order to keep the language simpler. The subset is shown in Figure 2.1.
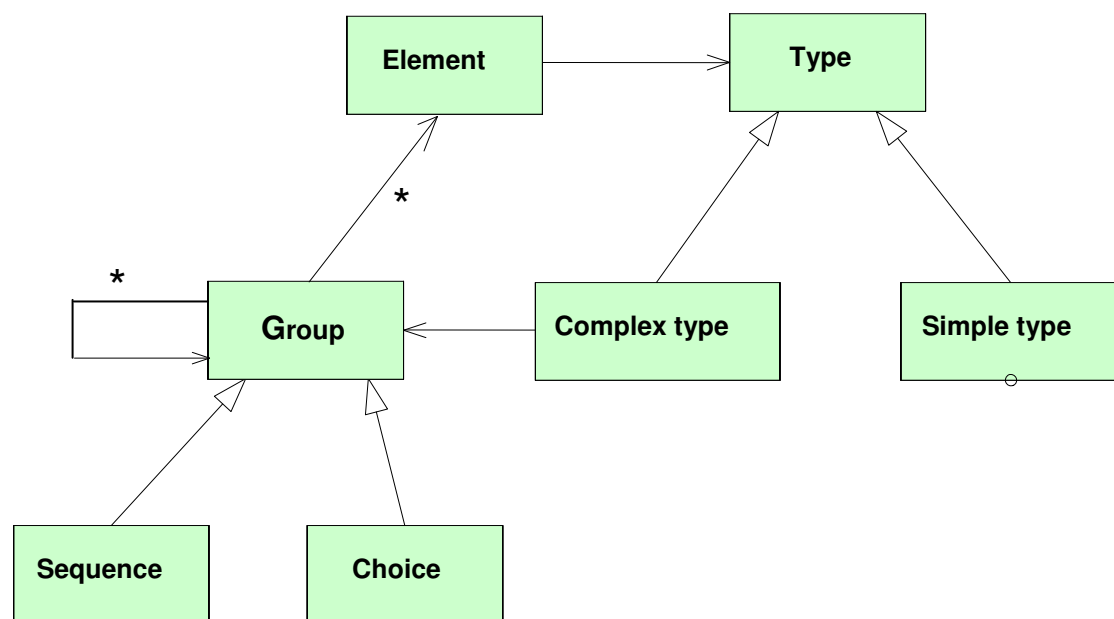

Fig 2.1. DFDL Schema subset

This subset is enough to model the logical structure of all kinds of non-XML data.

Consider the simple Address example from lesson 1 where we have an address that consists of a number of variable length fields.

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

Each individual field is modeled by an *element*. This supplies the name of the field, the number of occurrences of the field, and refers to a *type* that describes the field content.

If a field is a value, then it is modeled by a *simple type*, for example, an integer or a string. XML Schema provides a set of built-in simple types and DFDL allows a subset of those. You can also create your own, by deriving from the built-in types.

If a field has structure, then it is modeled by a *complex type*, which contains a *group*. The group defines the content for the complex type, by referring to the elements that model the individual fields that make up the structure. A group can either be a *sequence* or a *choice*. A sequence is used when all the fields in the structure appear (they can be ordered or unordered). A choice is used when only one of the fields in the structure may appear (that is they are alternatives). A group can directly contain another group but that's a more advanced feature.

When an element is referred to in a group, you can specify whether the element is optional, mandatory or repeats.

Note that a field may have either a value or structure, it can not have both. Put in DFDL terms, an element may have either a simple type or a complex type, but not both.

In the example, we can immediately see fields for house number, street, city and state. These are all fields that have a value, and therefore a Simple Type. But there's also a field that's not so obvious; the overall address itself. This field has structure, and therefore a Complex Type.

One way of modeling the example in XML Schema looks like this:

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">

  <xs:element name="address" type="addressType"/>

  <xs:complexType name="addressType">
    <xs:sequence>
      <xs:element ref="houseNumber"/>
      <xs:element ref="street"/>
      <xs:element ref="city"/>
      <xs:element ref="state"/>
    </xs:sequence>
  </xs:complexType>

  <xs:element name="houseNumber" type="xs:int"/>

  <xs:element name="street" type="xs:string"/>

  <xs:element name="city" type="xs:string"/>

  <xs:element name="state" type="xs:string"/>
```

```
</xs:schema>
```

In this model, all the elements and the complex type are given a name and are *global* allowing them to be re-used. For example, you could go on to create another complex type that also referred to the "houseNumber" element. Global objects are great for re-use but you can create a more compact model by making objects *local*. This is the recommended approach if you know particular objects will not be re-used. Here's the same model but using local objects.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">

  <xs:element name="address">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="houseNumber" type="xs:int"/>
        <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>
```

DFDL's XML Schema subset also permits the use of *namespaces*. This is a useful mechanism if you want to make sure that your DFDL model does not clash with a model belonging to someone else. When using namespaces, the name of each global object in the schema is qualified by the namespace of its schema. Here's the same model but using a namespace.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
           targetNamespace="http://myAddress.com"
           xmlns:addr="http://myAddress.com" >

  <xs:element name="address">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="houseNumber" type="xs:int"/>
        <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>
```

Any subsequent references to global element "address" are qualified by the namespace *prefix* "addr" declared at the top of the schema.

```xml
  <xs:element ref="addr:address" />
```

DFDL annotations

Using DFDL's subset of XML Schema allows you to describe the logical structure of your data. We will now look at how DFDL annotations allow the physical representation of your data to be described.

An XML Schema annotation provides some extra information about an XML Schema object such as an element, type or group. There are two kinds of annotation, *documentation* and *appinfo*. While documentation is intended to provide humans with a nice readable description for the object, appinfo is intended for programs. With DFDL, you use appinfo annotations to describe the physical representation of your data.

Here's what a DFDL appinfo annotation looks like.

```
<xs:annotation>
  <xs:appinfo source="http://www.ogf.org/dfdl/" >
    ..physical representation description goes here..
  </xs:appinfo>
</xs:annotation>
```

The source attribute says that this appinfo belongs to DFDL, so other processors know to ignore it.

The actual physical representation description takes the form of special DFDL XML elements that carry DFDL *properties* that describe the physical data. There is a special DFDL element for each kind of DFDL Schema object for which properties are provided, as shown in table 2.1.

Annotation	Used on Component	Purpose
dfdl:element	xs:element xs:element reference	Contains the DFDL properties of a xs:element and xs:element reference
dfdl:choice	xs:choice	Contains the DFDL properties of a xs:choice.
dfdl:sequence	xs:sequence	Contains the DFDL properties of a xs:sequence.
dfdl:group	xs:group reference	Contains the DFDL properties of a xs:group reference to a group definition containing an xs:sequence or xs:choice.
dfdl:simpleType	xs:simpleType	Contains the DFDL properties of a xs:simpleType

Table 2.1. Core DFDL annotation elements

Note that xs:complexType does not carry a DFDL annotation.

This is what the houseNumber element from the variable length Address example might look like once it has a DFDL annotation added. In English, this is saying that 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>
```

If houseNumber was instead a big-endian 4 byte fixed length two's complement binary integer then the DFDL annotation might look like this.

```
<xs:element name="houseNumber" type="xs:int">
  <xs:annotation>
    <xs:appinfo source="http://www.ogf.org/dfdl/" >
      <dfdl:element representation="binary"
                    binaryNumberRep="binary"
                    lengthKind="explicit"
                    length="4"
                    lengthUnits="bytes"
                    byteOrder="bigEndian" .../>
    </xs:appinfo>
  </xs:annotation>
</xs:element>
```

The DFDL annotations shown above are known as *attribute form*. A more compact equivalent syntax is also provided known as *short form*, which is explained in lesson 3.

The number of DFDL properties is very large, providing the ability to model a huge variety of data formats. These are fully described in the DFDL specification, and many are further described in later lessons.

DFDL provides further annotations that enable you to:
  o Set up groups of DFDL properties and have them apply to all objects in a DFDL Schema;
  o Make assertions about objects;
  o Resolve points of uncertainty in the model;
  o Define escape schemes for text data;
  o Define, instantiate and set variables.

These are briefly described in table 2.2, and described in more detail in later lessons.

Annotation	Used on Component	Purpose
dfdl:format	xs:schema dfdl:defineFormat	Contains a set of DFDL properties that can be used by multiple DFDL schema objects. When dfdl:format is used directly on the xs:schema itself, the property values act as defaults for all objects in the DFDL schema.
dfdl:defineFormat	xs:schema	Defines a reusable data format by associating a name with a set of DFDL properties contained within a child dfdl:format annotation. The name can be referenced from DFDL annotations on multiple DFDL schema objects, using dfdl:ref.
dfdl:assert	xs:element, xs:choice xs:sequence, xs:group	Defines a test to be used to ensure the data are well formed. Used only when parsing data.
dfdl:discriminator	xs:element, xs:choice xs:sequence, xs:group	Defines a test to be used when resolving a point of uncertainty such as choice branches or optional elements. Used only when parsing data.
dfdl:escapeScheme	dfdl:defineEscape Scheme	Defines a scheme by which quotation marks and escape characters can be specified. This is for use with text formats.
dfdl:defineEscape Scheme	xs:schema	Defines a reusable escape scheme by associating a name with a contained child dfdl:escapeScheme annotation. The name can be referenced from DFDL annotations on multiple DFDL schema objects.
dfdl:defineVariable	xs:schema	Defines a variable that can be referenced elsewhere. This can be used to communicate a parameter from one part of processing to another part.
dfdl:newVariable Instance	xs:element, xs:choice xs:sequence, xs:group	Creates a new instance of a variable.
dfdl:setVariable	xs:element, xs:choice xs:sequence, xs:group	Sets the value of a variable whose declaration is in scope.

Table 2.2. Other DFDL annotations

Reusing DFDL schemas

Let's assume you have created a DFDL schema for the address element, in a file called address.xsd. You now want to use that address element in another DFDL schema you are creating called customer.xsd. One approach is to take the address element and copy it into customer.xsd. A better approach is to use XML Schema's *include* or *import* capability, both of which are part of DFDL's XML Schema subset. This enables the creation of modular models that cater for reuse.

Use *include* when either namespaces are not involved or the two schemas have the same namespace.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">

  <xs:include schemaLocation="address.xsd"/>

  <xs:element name="customer">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="firstName" type="xs:string"/>
        <xs:element name="surname" type="xs:string"/>
        <xs:element ref="address"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>

</xs:schema>
```

Use *import* when the two schemas have different namespaces.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
           targetNamespace="http://myCustomer.com"
           xmlns:addr="http://myAddress.com"
           xmlns:cust="http://myCustomer.com" >

  <xs:import schemaLocation="address.xsd"/>

  <xs:element name="customer">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="firstName" type="xs:string"/>
        <xs:element name="surname" type="xs:string"/>
        <xs:element ref="addr:address"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>

</xs:schema>
```