DFDL Training: The FakeTDL Data Format

A Hands-on Training Class Using

APACHE
Daffodil™

Assumptions - Prerequisites

- Seen the DFDL Overview Presentation
- Know a bit of XML
  - w3schools XML Tutorial - basic introduction to XML.
  - Our Slides: Introduction to XML
- Know a bit of XML Schema (aka XSD)
  - w3schools XML Schema Tutorial - basics about XSD.
  - Our Slides: Introduction to XML Schema

Agenda

- A data format for learning DFDL - FakeTDL

- Hands-on Labs
  - Create/debug and Improve a DFDL Schema
  - How to structure a schema project
  - Incorporate testing
  - Tools: Daffodil command line, SBT build tool

Goals of DFDL Training

- Create/Review an interesting example DFDL Schema

- Learn
  - DFDL properties that are needed
  - Common DFDL concepts and terminology
  - How to structure and test a DFDL Schema *before* deploying

- Learn how to self-teach about DFDL
  - What are the sources of information?
  - How to find things in the DFDL Spec
  - Where to get help
  - Where are more training materials

FakeTDL

A data format for learning DFDL

FakeTDL - Our Example Data Format

- A completely fictional data format
- Some similarities to Tactical Data Link (TDL) data.
  - Geolocations - lat/lon/elevation are common
  - Track/Unit identifier strings - ex: "AG147"
  - Fixed length fields
  - Binary data mostly. A few fields are strings/chars.
- Other characteristics
  - Byte oriented (nothing smaller than 1 byte - no bits)
  - Most field types have typical sizes implied by their types
    - unsignedInt is 4 bytes,
    - short is 2 bytes,
    - float is 4 bytes
  - Big endian byte order
  - ASCII for the few fields that are text
  - Unused bytes contain 0x53 (which is character 'X' in ASCII)
  - 64 bytes - complete message length

FakeTDL - has a Spec

- HTML and PDF versions
  - FakeTDLSpecification.html

- Highlights to look at are sections:
  - FakeTDL Message Details
    - general nature of the format
  - Track Message Fields
    - look at each field's length and type

FakeTDL Format Basics

FakeTDL Spec	DFDL Properties

binary (not text)	representation="binary"
	binaryNumberRep="binary"

big endian	byteOrder="bigEndian"
ascii	encoding="ascii"

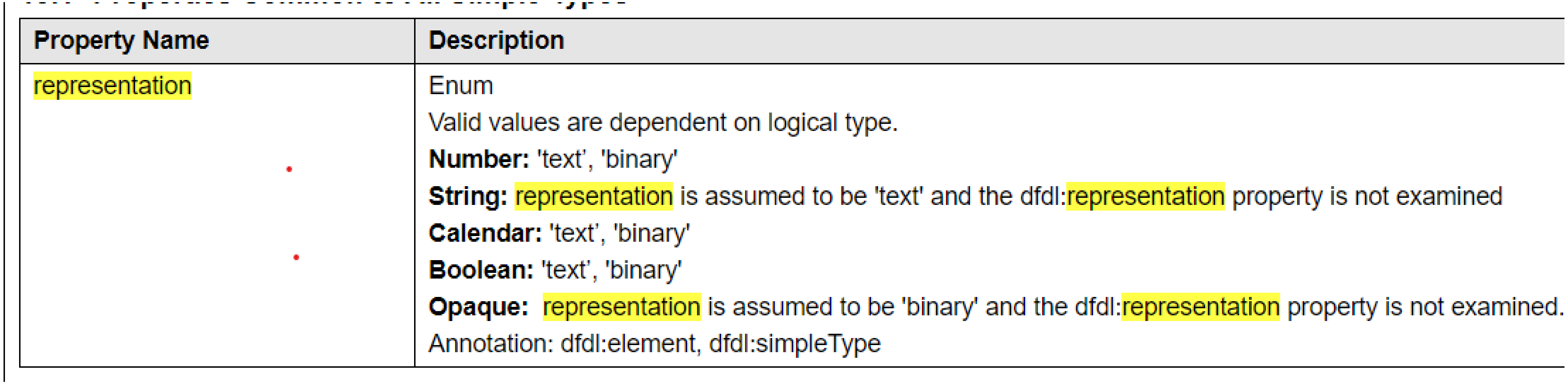
fixed length messages	lengthKind="explicit"
	length="64"

byte centric (nothing smaller than a byte)	alignment="1"
	alignmentUnits="bytes"
	lengthUnits="bytes"

typical field sizes ex: short is 2 bytes float is 4 bytes	lengthKind="implicit"
unused bytes filled with 'X'	fillByte='X'

Finding Properties in the DFDL Spec

- Lookup properties in the DFDL Spec.
- Use search (there is no index by property name)
- Search until you find the *property box:*



Property Name	Description
representation	Enum Valid values are dependent on logical type. **Number:** 'text', 'binary' **String:** representation is assumed to be 'text' and the dfdl:representation property is not examined **Calendar:** 'text', 'binary' **Boolean:** 'text', 'binary' **Opaque:** representation is assumed to be 'binary' and the dfdl:representation property is not examined. Annotation: dfdl:element, dfdl:simpleType

- After the table there are often sections that elaborate on the properties in that table.

DFDL Spec Overview

- Details - skip over for now
  - Section 3, Notational and Definitional Conventions - also Appendix E: Glossary of Terms.
  - Section 4, The DFDL Information Set (Infoset)
  - Section 5, DFDL Schema Component Model
  - Sections 6, DFDL Syntax Basics and 7, Syntax of DFDL Annotation Elements
  - Section 8, Property Scoping and DFDL Schema Checking

- Section 9, DFDL Processing Introduction
  - DFDL Data Syntax Grammar
  - Parsing Algorithm - Points of Uncertainty

✓ Section 10, Overview: Representation Properties and their Format Semantics
  - Common to both Content and Framing (see Section 11)
  - Common Framing, Position, and Length (see Section 12)
  - Simple Type Content (see Section 13) - Biggest section - text and binary properties for all types
  - Sequence Groups (see Section 14)
  - Choice Groups (see Section 15)
  - Array (i.e., recurring) elements and optional elements (see Section 16)
  - Calculated Values (see Section 17)

- Details - use when needed
  - Section 18, DFDL Expression Language
  - Section 19, DFDL Regular Expressions

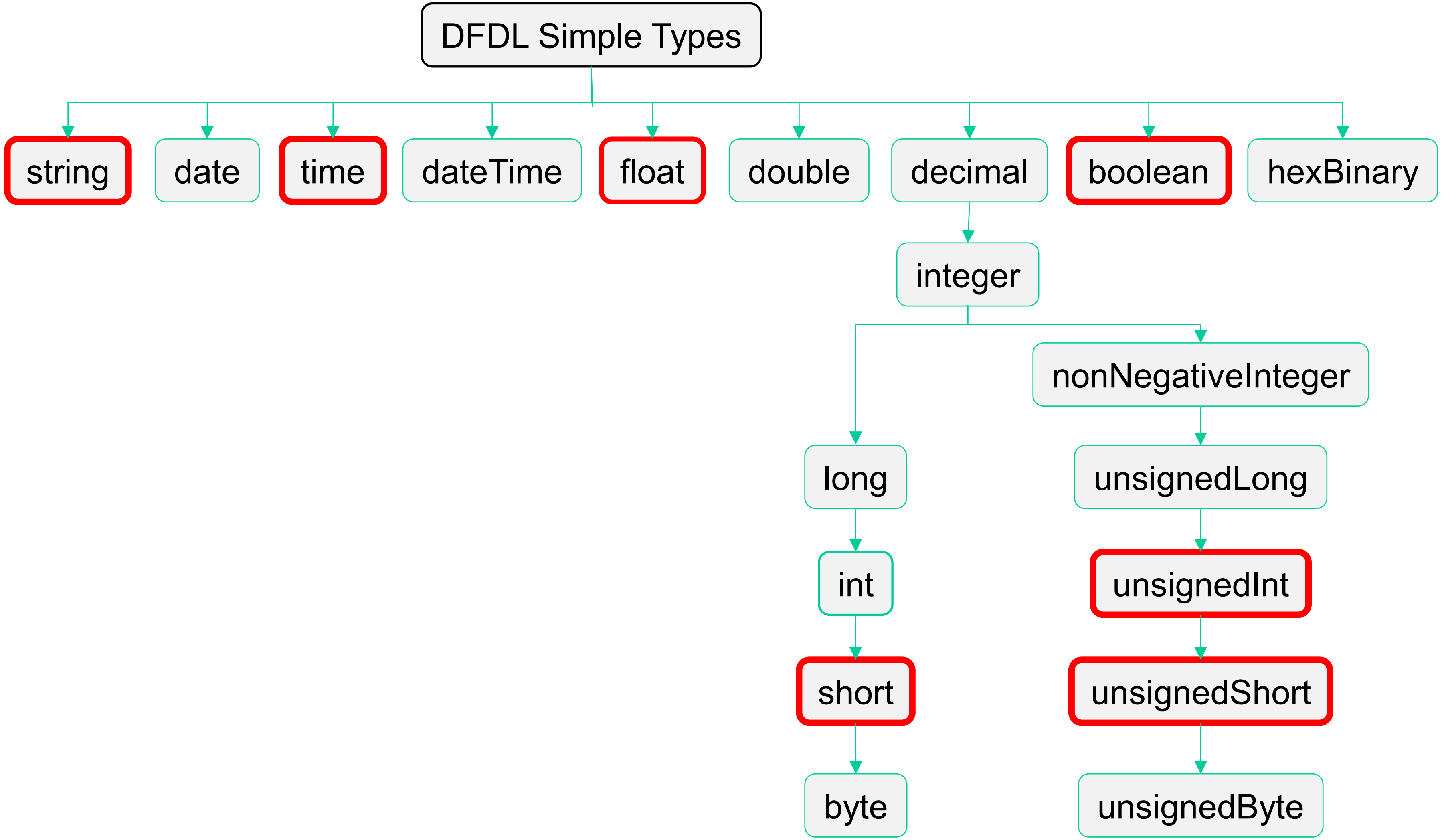
dfdl:format

- The format that applies to everything in the DFDL schema file

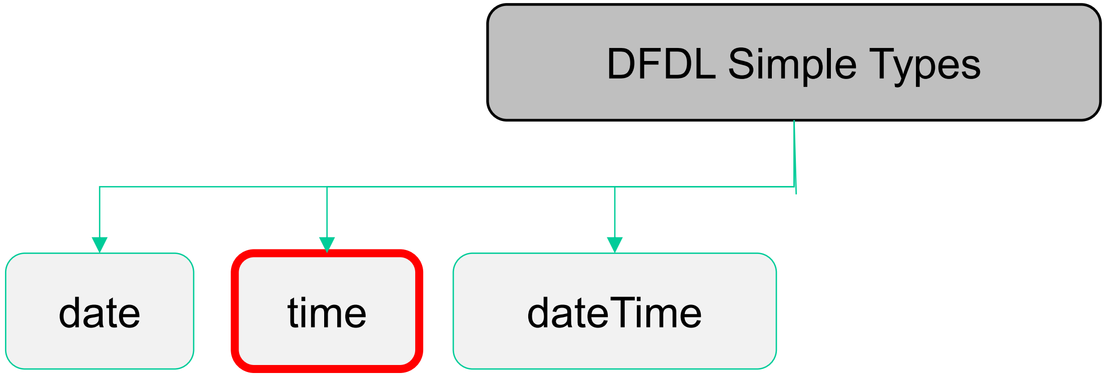
```
<dfdl:format ...
  representation="binary"
  binaryNumberRep="binary"
  byteOrder"bigEndian"
  lengthKind"implicit"
  lengthUnits="bytes"
  alignmentUnits="bytes"
  alignment="1"
  encoding="ascii"
  fillByte'X' />
```

What we mean
by *byte centric*

DFDL Types and Used by FakeTDL



Terminology: *Calendar* types



DFDL Spec Sections

- generally
  - 12.3 Properties for Specifying Length
- string
  - 13.4 Properties Specific to String
- short, unsignedShort, unsignedInt
  - 13.7 Properties Specific to Number with Binary Representation
- float
  - 13.8 Properties Specific to Float/Double with Binary Representation
- boolean
  - 13.10 Properties Specific to Boolean with Binary Representation
- time
  - 13.11 Properties Specific to Calendar with Text or Binary Representation
  - 13.13 Properties Specific to Calendar with Binary Representation

FakeTDL Message Fields

- All messages start with these 3 fields
  - messageType - 1 char/byte
  - source unit number - 5 chars/bytes
  - message send time - 3 bytes *Binary Coded Decimal (BCD)*

- 3 Kinds of messages
  - ✓ Track
  - Identity
  - Ack

FakeTDL Track Message as XML

```
<fakeTDL>
  <track>                                 <!-- 1 bytes  initiator 'T'   -->
    <source>AG123</source>                <!-- 5 bytes  string          -->
    <sendTime>01:02:03</sendTime>         <!-- 3 bytes  time            -->
    <mustAck>false</mustAck>              <!-- 1 byte   boolean         -->
    <messageID>1</messageID>              <!-- 4 bytes  unsignedInt     -->
    <sourceLat>41.0</sourceLat>           <!-- 4 bytes  float           -->
    <sourceLon>-70.0</sourceLon>          <!-- 4 bytes  float           -->
    <sourceElev>400</sourceElev>          <!-- 2 bytes  short           -->
    <trackNum>UU777</trackNum>            <!-- 5 bytes  string          -->
    <time>01:02:01</time>                 <!-- 3 bytes  time            -->
    <lat>41.1</lat>                       <!-- 4 bytes  float           -->
    <lon>-69.9</lon>                      <!-- 4 bytes  float           -->
    <elev>350</elev>                      <!-- 2 bytes  short           -->
    <pointType>W</pointType>              <!-- 1 byte   string          -->
    <quality>A</quality>                  <!-- 1 byte   string          -->
    <course>75</course>                   <!-- 2 bytes  unsignedShort -->
    <speed>200</speed>                    <!-- 2 bytes  unsignedShort -->
  </track>
</fakeTDL>
```

Two Unit + Time Pairs

```
<fakeTDL>
  <track>                              <!-- 1 bytes  initiator 'T'  -->
    <source>AG123</source>             <!-- 5 bytes  string         -->
    <sendTime>01:02:03</sendTime>      <!-- 3 bytes  time           -->
    <mustAck>false</mustAck>           <!-- 1 byte   boolean        -->
    <messageID>1</messageID>           <!-- 4 bytes  unsignedInt -->
    <sourceLat>41.0</sourceLat>        <!-- 4 bytes  float          -->
    <sourceLon>-70.0</sourceLon>       <!-- 4 bytes  float          -->
    <sourceElev>400</sourceElev>       <!-- 2 bytes  short          -->
    <trackNum>UU777</trackNum>         <!-- 5 bytes  string         -->
    <time>01:02:01</time>              <!-- 3 bytes  time           -->
    <lat>41.1</lat>                    <!-- 4 bytes  float          -->
    <lon>-69.9</lon>                   <!-- 4 bytes  float          -->
    <elev>350</elev>                   <!-- 2 bytes  short          -->
    <pointType>W</pointType>           <!-- 1 byte   string         -->
    <quality>A</quality>               <!-- 1 byte   string         -->
    <course>75</course>                <!-- 2 bytes  unsignedShort -->
    <speed>200</speed>                 <!-- 2 bytes  unsignedShort -->
  </track>
</fakeTDL>
```

Two Geolocation Triples

```
<fakeTDL>
  <track>
    <source>AG123</source>                       <!-- 5 bytes  string        -->
    <sendTime>01:02:03</sendTime>                <!-- 3 bytes  time          -->
    <mustAck>false</mustAck>                     <!-- 1 byte   boolean       -->
    <messageID>1</messageID>                     <!-- 4 bytes  unsignedInt -->
    <sourceLat>41.0</sourceLat>                  <!-- 4 bytes  float         -->
    <sourceLon>-70.0</sourceLon>                 <!-- 4 bytes  float         -->
    <sourceElev>400</sourceElev>                 <!-- 2 bytes  short         -->
    <trackNum>UU777</trackNum>                   <!-- 5 bytes  string        -->
    <time>01:02:01</time>                        <!-- 3 bytes  time          -->
    <lat>41.1</lat>                              <!-- 4 bytes  float         -->
    <lon>-69.9</lon>                             <!-- 4 bytes  float         -->
    <elev>350</elev>                             <!-- 2 bytes  short         -->
    <pointType>W</pointType>                     <!-- 1 byte   string        -->
    <quality>A</quality>                         <!-- 1 byte   string        -->
    <course>75</course>                          <!-- 2 bytes  unsignedShort -->
    <speed>200</speed>                           <!-- 2 bytes  unsignedShort -->
  </track>
</fakeTDL>
```

A Few Misc Fields

```
<fakeTDL>
  <track>
    <source>AG123</source>                  <!-- 5 bytes  string        -->
    <sendTime>01:02:03</sendTime>           <!-- 3 bytes  time          -->
    <mustAck>false</mustAck>                <!-- 1 byte   boolean       -->
    <messageID>1</messageID>                <!-- 4 bytes  unsignedInt -->
    <sourceLat>41.0</sourceLat>             <!-- 4 bytes  float         -->
    <sourceLon>-70.0</sourceLon>            <!-- 4 bytes  float         -->
    <sourceElev>400</sourceElev>            <!-- 2 bytes  short         -->
    <trackNum>UU777</trackNum>              <!-- 5 bytes  string        -->
    <time>01:02:01</time>                             <!-- 3 bytes  time          -->
    <lat>41.1</lat>                         <!-- 4 bytes  float         -->
    <lon>-69.9</lon>                        <!-- 4 bytes  float         -->
    <elev>350</elev>                        <!-- 2 bytes  short         -->
    <pointType>W</pointType>                <!-- 1 byte   string        -->
    <quality>A</quality>                    <!-- 1 byte   string        -->
    <course>75</course>                     <!-- 2 bytes  unsignedShort -->
    <speed>200</speed>                      <!-- 2 bytes  unsignedShort -->
  </track>
</fakeTDL>
```

Track Message

(from xxd test_track_good_01.dat)

```
00000000: 5441 4731 3233 0102 0300 0000 0001 4224  TAG123........B$
00000010: 0000 c28c 0000 0190 5555 3737 3701 0201  ........UU777...
00000020: 4224 6666 c28b cccd 015e 5741 004b 00c8  B$ff.....^WA.K..
00000030: 5858 5858 5858 5858 5858 5858 5858 5858  XXXXXXXXXXXXXXXX
```

```
<source>AG123</source>
<sendTime>01:02:03</sendTime>
...
```

```
<mustAck>false</mustAck>
<messageID>1</messageID>
<sourceLat>41.0</sourceLat>
<sourceLon>-70.0</sourceLon>
<sourceElev>400</sourceElev>
<trackNum>UU777</trackNum>
<time>01:02:01</time>
<lat>41.1</lat>
<lon>-69.9</lon>
<elev>350</elev>
<pointType>W</pointType>
<quality>A</quality>
<course>75</course>
<speed>200</speed>
```

DFDL Property binaryCalendarRep

- Time 01:02:03 (2 mins and 3 seconds after 1am)
- BCD – binary coded decimal
  - Data in hex: 0x01, 0x02, 0x03.
- One hex digit == one decimal digit.
- Almost like text, but only 4 bits per digit.
- BCD is often used for decimal numbers representing money. Also common for dates/times.

Lab Exercises - Hands On

	Topic(s)		
01	Track message schema	• Fill in missing parts of a schema for Track messages. • Study DFDL properties. • Learn to find things in the DFDL Spec. • Use Daffodil CLI to parse/unparse data to/from XML	• Encounter different kinds of errors (SDE, PE)
02	Add built-in tests	• Use Test Data Markup Language (TDML) to create test suite built into the schema project • Run TDML tests from the Daffodil CLI	
03	Improve schema	• Add types with facets to satisfy XML ISG guidance • Well-formed and Valid vs. Malformed • Reusable types, better schema organization • LengthKind 'implicit' for strings • ISG for XSD rules	
04	Finish schema: Add Identity and Ack messages	• Add choice of Track/Ident/Ack messages to schema • Arrays in Ack message with Stored Count • discriminators	
05	Production and Maintainability	• Use daffodil-sbt to compile schema • Add test JUnit drivers: 'sbt test' runs all tests. • Eliminate namespace prefixes. • Test files of messages, not just individual messages.	

Lab01

Track Message, First Cut

Lab01 - a First Cut Schema

File name	Purpose/Role
README.md	Explanation and some command lines to try
fakeTDL.dfdl.xsd	The DFDL Schema for the format Partial - We will fill in missing fields and properties.
test_track_good_01.dat	Data file with good track message
test_track_good_01.xml	Expected result of parsing test_track_good_01.dat
test_track_bad_01.dat	Data file with bad track message (too short)
test_track_bad_02.xml	XML file with bad infoset values (for unparse test)

Using the Daffodil Command Line Interface (CLI)
See: https://daffodil.apache.org/cli/

fakeTDL.dfdl.xsd

- ".dfdl.xsd" file name convention for DFDL schemas
- default namespace is XMLSchema
  - avoid having to type/read "xs:" in front of all the keywords
- target namespace
  - "fakeTDL:" namespace prefix
- dfdl:format extends from DFDLGeneralFormat
  - applies to the whole file
- fakeTDL - global root element (one liner)
- trackMessageType - contains fields of track msg

Review of DFDL Properties from Lab01

- initiator
- lengthKind 'explicit' (Section 12.3.1)
- length
- binaryCalendarRep
- calendarPattern
- calendarPatternKind
- lengthKind 'implicit' (Section 12.3.3)

Review: Different Kinds of Errors

- Schema Definition Error
  - the DFDL schema has an error – so it is not meaningful
  - detected at schema compilation time (before parse/unparse begins)
- Parse Error
  - the data has an error or doesn't match the schema
  - causes backtracking
    - try other choice alternatives
    - optional elements/variable-length array
  - only fatal if there are no alternatives for the parser to try
- Unparse Error
  - always fatal - unparsing fails

Lab02 - TEST/QA for DFDL Schemas

DFDL Schema Built-in-Self-Test (BIST)

using Test Data Markup Language (TDML)

Built-In TDML Self Test

- Essential to test and debug the DFDL schema before deployment
- Debug of problems easier "off box"
- Every DFDL Schema project should have built-in testing
- Easy & uniform means to add new tests

Test Data Markup Language (TDML)

- XML syntax for writing (and managing) DFDL tests
  - parserTestCase
  - unparserTestCase
  - Test cases can be positive or negative (expect errors of various kinds)

- A TDML file glues together
  - DFDL schema
  - test data
  - test infoset (XML)
  - list of expected warnings
  - list of expected errors (for negative tests)

- TDML Doc Link: https://daffodil.apache.org/tdml/
- XML Schema for TDML:
  - https://s.apache.org/daffodil-tdml.xsd

Lab02 - Using TDML

- Run the test suite of positive and negative test cases using the daffodil CLI

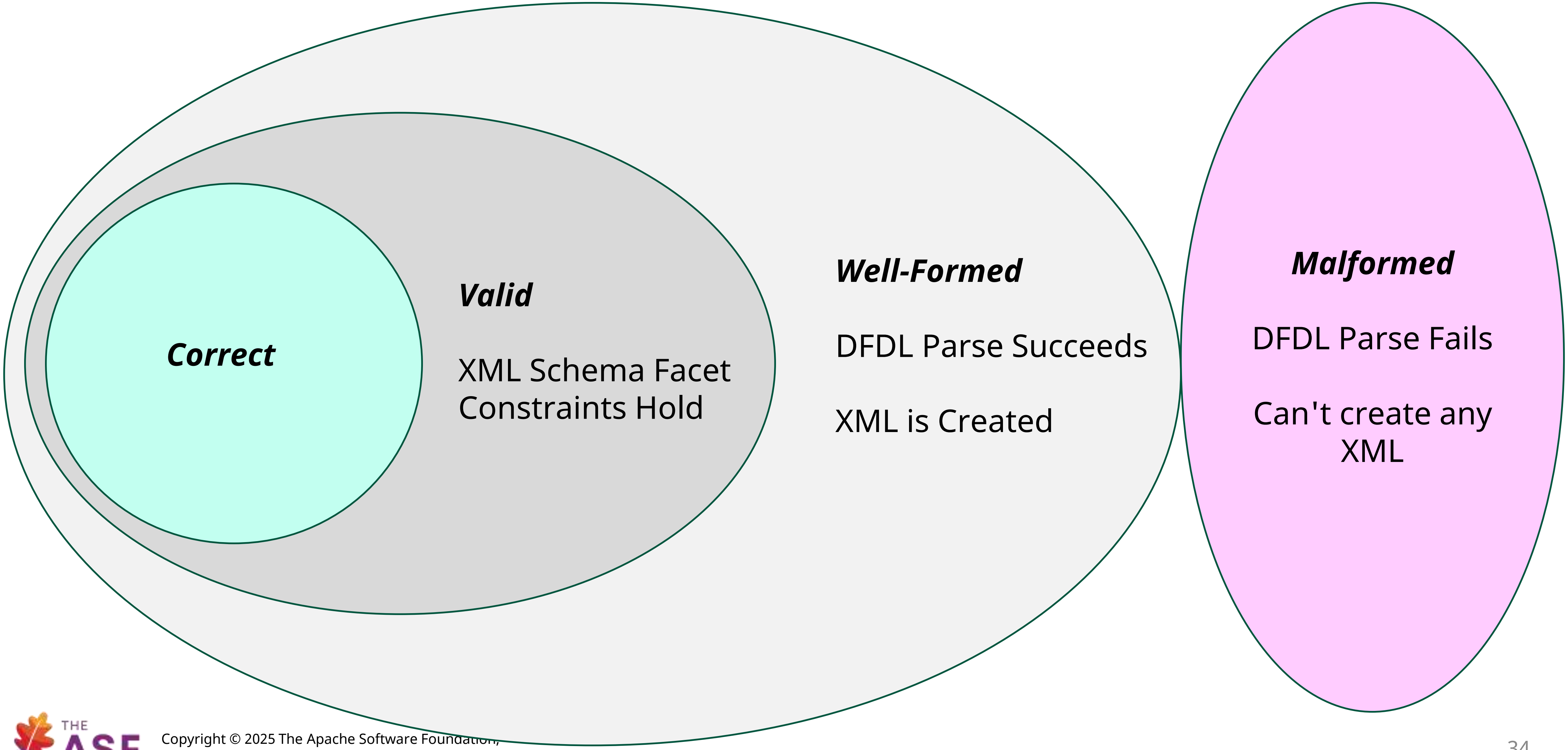
- A quick lab.
  - Just review and try out TDML testing.

- We will add more tests as we enhance the schema

Improving the Schema

Adding validity checking.

Data Quality Concepts



Correct

Valid

XML Schema Facet Constraints Hold

Well-Formed

DFDL Parse Succeeds

XML is Created

Malformed

DFDL Parse Fails

Can't create any XML

Data Quality Concepts

- Malformed Data
  - DFDL Parse fails - we cannot even create XML from the data

- Well-Formed Data
  - Can find every field's location and length
  - Can convert each field to its logical type
  - DFDL Parse can succeed
  - Can create XML from the data
  - But note: ***This XML may not be valid***

- Valid Data
  - Obeys schema constraints (facets)
    - Range of numbers, dates, times, patterns of text
  - Validation usually done by separate filter step, not the DFDL Parser

- Correct Data
  - Works in all systems/cases

Our DFDL Schema so far…

- Will NOT pass serious scrutiny. Why?

- It must be a good DFDL schema
  - parse *well-formed* data
    - ✓ Yes
  - reject *malformed* data
    - ✓ Yes: test_track_bad_02 rejected the malformed time "26:99"

- It also must be a good *XML schema*
  - tight validity constraints (facets)
  - identify *invalid* data

FakeTDL pointType Field

- Spec says
  - 1 character
  - S, W, E - for start, waypoint, or end of track

- As of Lab02, the schema has only:

```
<element name="pointType" type="xs:string"
    dfdl:lengthKind="explicit"
    dfdl:length="1">
```

FakeTDL pointType Field

- Improved schema will have…

```
<element name="pointType" type="fakeTDL:trackPointType"/>

<simpleType name="trackPointType"
  dfdl:lengthKind="explicit" dfdl:length="1">
  <restriction base="xs:string">
    <minLength value="1"/>
    <maxLength value="1"/>
    <enumeration value="S"/>
    <enumeration value="W"/>
    <enumeration value="E"/>
  </restriction>
</simpleType>
```

XSD "Facets"

FakeTDL pointType Field

- minor additional improvement
  - removes a bit of redundancy

```
<simpleType name="pointType" dfdl:lengthKind="implicit">
  <restriction base="xs:string">
    <minLength value="1"/>
    <maxLength value="1"/>
    <enumeration value="S"/>
    <enumeration value="W"/>
    <enumeration value="E"/>
  </restriction>
</simpleType>
```

Lab03

Improving the Track Message Schema

Adding Types with Validation Facets

Lab03 exercises

- Change schema to define simple types with facets for all fields.
  - Test on well-formed, but invalid data, to get validation errors
  - Modify test suite to expect validation errors

DFDL Properties and XSD Facets

- XSD simpleType definitions
  - referenced from elements
  - restrictions added to base type
  - units of measure - naming convention
    - latitude_degrees, elevation_25FeetMSL
  - share common definitions
- XSD facets
  - minLength, maxLength - strings
  - pattern (a regular expression)
  - minInclusive, maxInclusive - numbers
  - enumeration - any simple type

Multiple Message Types

Arrays, Choices, Points of Uncertainty, Discriminators

Testing Files of Messages

This section…

- Add FakeTDL Identity and Ack messages
  - Regex XSD ISG guidance
  - Arrays with stored counts
    - dfdl:occursCountKind 'expression'
- Choice
  - dfdl:initiatedContent="yes" discriminates choice
- Arrays - multiple messages in a file
  - DFDL Discriminators on arrays

Lab04

Choice of more Message Types, Arrays

Lab04

- adds new messages
  - uses a choice, dfdl:initiatedContent and dfdl:initiator to choose which message
- ack contains an array with *stored count* field
  - uses dfdl:occursCount expression and dfdl:outputValueCalc expression to use the count field and ensure it is unparsed properly

Lab04 - Added New Messages

- Identity and Ack

- All messages have dfdl:initiator

- Property dfdl:initiatedContent="yes"
  - makes the initiator into a *discriminator*
  - choice *Point of Uncertainty* (PoU) is resolved by finding the dfdl:initiator

- See DFDL Spec Section 9.3
  - One of the most complex aspects of the DFDL spec.

Choice with dfdl:initiatedContent

- contrast choice with dfdl:initiatedContent="yes" vs. no
    - for negative test case (malformed BCD sendTime)
    - diagnostic is misleading without it.
- Good example of getting proper diagnostic for negative test cases

Identity Message

- simpleType entityTypeDetail
  - example of numeric enum with many entries
  - only a few are shown. They can be big (hundreds) in real schemas.

- Daffodil has extensions to DFDL to do enums better.
  - See the DFDLSchemas MIL-STD-2045 schema on github for examples and usage.

Identity Message

- simpleType identDescription
  - dfdl:textTrimKind 'padChar'
  - dfdl:textPadKind 'padChar'
  - dfdl:textStringPadCharacter '%#r00;'
    - a DFDL raw byte entity (for ASCII NUL)
  - dfdl:textStringJustification 'left'
    - padding characters trimmed/added to right
  - fairly complex pattern regex needed
  - Note: avoids use of "*" and "+" regex
    - to conform with ISG guidance on secure XSD

Fill, Pad, and Trim Properties

- DFDL uses the term "Fill" for unused parts of the data format.
  - dfdl:fillByte value is used to fill things in.

- DFDL uses the term "Pad" and "Trim" for text fields with characters before, after, or around the data.
  - Trimming happens when parsing
  - Padding happens when unparsing
  - Properties
    - textTrimKind textPadKind
  - The padding character to trim or add:
    - textStringPadCharacter, textNumberPadCharacter, textBooleanPadCharacter or textCalendarPadCharacter
  - Where padding is trimmed/added:
    - textStringJustification, textNumberJustification, textBooleanJustification, textCalendarJustification
    - Numbers are typically right justified. Other things left justified.

- Not every format spec is consistent with DFDL's terminology on pad vs. fill.

Lab04 - Ack Message & Arrays

- Contains an array of messageID items
- DFDL Array & Optional Properties
  - dfdl:occursCountKind property
  - dfdl:occursCount property
- Count is stored in the data
  - Count is used by dfdl:occursCount expression
  - Recomputed on unparsing via dfdl:outputValueCalc

- DFDL terminology
  - Array: 0 to 2 or more are *possible*
  - Optional: 0 or 1 only
  - Scalar: Exactly 1 only
  - dfdl:occursCountKind applies to Array and Optional elements
    - ignored for scalar elements

Summary: Different Kinds of Errors

- Schema Definition Error
  - the DFDL schema has an error
  - usually detected at schema compilation time (before parse/unparse begins)
- Parse Error
  - the data has an error or doesn't match the schema
  - causes backtracking to try other choice alternatives
  - causes optional elements/variable-length array elements to stop parsing more elements
  - only fatal if there are no alternatives for the parser to try
- Unparse Error
  - always fatal - unparsing fails
- Validation Error
  - if Daffodil is run with validation options selected
  - These do not cause backtracking
- Left-over data
  - parse succeeded, but did not consume all the data
- TDML negative tests can expect any of these

Lab05

Production Schema Organization

Using SBT and the daffodil-sbt Plugin

Using SBT to Simplify Build & Test

- build.sbt - defines your schema 'project'
- project directory
  - build.properties - sbt version to use
  - plugins.sbt
    - specifies daffodil-sbt plugin
    - future: other plugins
- TestFakeTDL.scala - JUnit test driver
  - enables running tests easily
    - from command line
    - from IDE
      - JetBrains IDEA, VSCode both work well for DFDL schema work
      - There's a Daffodil VSCode extension in the works to help write, debug DFDL schemas
- 'sbt test'
  - runs all JUnit tests

SBT daffodil-sbt Plugin

- Prepares DFDL schema for deployment

- 'sbt packageDaffodilBin'
  - Creates pre-compiled schema ".bin" files
  - For the specific Daffodil version you need
    - Depends on what Owl product and patch level
    - Could be Daffodil 3.5.0, 3.7.0, 3.8.0, 3.9.0, 3.10.0, 3.11.0, 4.0.0 ...
  - These ".bin" are used for parse and unparse.
    - The regular DFDL schema files (".dfdl.xsd") are used for separate XSD validation when required

End of DFDL Schema Labs !!!

You should have a great sense of accomplishment :-)

The result of lab05 is very similar to the official github fakeTDL schema.

Conclusion

Support/Help

- Free Support
  - Apache Daffodil project
    - join users@daffodil.apache.org mailing list
    - ask questions there - note: public archived list
    - email: daffodil-fouo-support@owlcyberdefense.com
      - non-public, but do not send FOUO/CUI materials
- Paid Support
  - Ask on users@daffodil.apache.org list

In Conclusion…

- Please provide feedback

END

That's all folks.

Extra or draft slides may follow this slide.