TRESYS
Deep.

DFDL and the Open Source Daffodil Project

presented to:

Anyone with Data

Agenda

- DFDL
  - Why is DFDL Needed?
  - Quick overview of DFDL
  - Use Cases for DFDL
- DFDL Tutorial
  - Examples: CSV, PCAP, MIL-STD-2045, USMTF*, VMF*
  - Java API
- Demonstrations
- DFDL Schemas
- Implementations
  - Open Source: Daffodil
- Conclusion

Why is DFDL Needed?

There are *hundreds* of ad-hoc data format description systems

Every Enterprise Software Company

- IBM (10+)
- Oracle(10+)
- SAP(10+)
- Microsoft
- SAS
- Informatica
- SyncSort
- AbInitio
- Pervasive
- Qlik/Expressor
- Pentaho
- Talend
- .... Dozens more

Every kind of software that takes in data:

- data directed routing (msg brokers)
- database
- data analysis and/or data mining
- data cleansing
- master data management
- application integration
- data visualization
- …

All these data format descriptions are:

- *proprietary*
- *ad-hoc*
- *incompatible*

Even within products of the same company!

TRESYS Deep.

Why DFDL is Needed?

Hundreds of data format description systems… means that:

- Vendor investment is spread too thin
  - Tools for creating data formats are inadequate
  - No product is comprehensive enough
  - Some products aren't fast enough
- Customers get locked in
- High training cost, low career value
- Inflexible packaging
  - not libraries - must embed some product in your application data flow

TRESYS Deep.

Why DFDL is Needed?

Result… we are stuck with:

- Data parsers continue to be written as programs
- High cost to write, high error rate.
- Certification (C&A) costs = $$$$$

A standard DFDL fixes all these problems

Data Format Description Language

- DFDL: A new open standard
  - From the Open Grid Forum (OGF)
  - Version 1.0 – Sept. 2014
  - Proposed Recommendation Status (as of Oct 2016)
- DFDL is a way of *describing* data…
  - It is NOT a data format itself!
- DFDL can describe …
  - Textual and binary
  - Commercial record-oriented
  - Scientific and numeric
  - Modern and legacy
  - Industry/Military standards
- While allowing high performance
  - You can choose the right data format for the job
- Leverage XML technology and concepts
  - Use W3C ***XML Schema subset*** & type system to describe the ***logical*** format of the data
  - Use annotations within the XSD to describe the ***physical*** representation of the data
- Read and write from same ***DFDL Schema***
- Keep simple cases simple
  - Simple DFDL Schemas are human readable

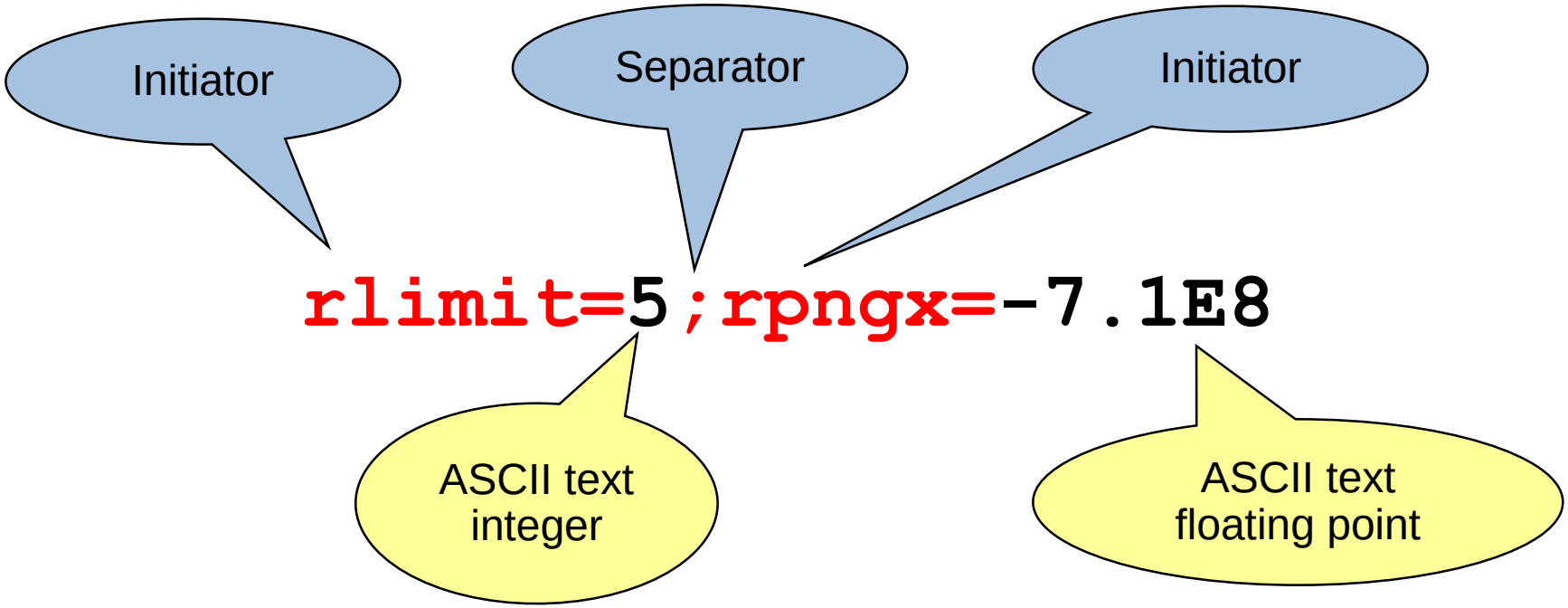
TRESYS Deep.

Example – Delimited Text Data

```
rlimit=5;rpngx=-7.1E8
```

TRESYS Deep.

Example – Delimited Text Data



Separators, initiators (aka tags), & terminators are all examples in DFDL of *delimiters*

TRESYS Deep.

DFDL Schema

```
<xs:complexType name="rValues">
  <xs:sequence>
    <xs:element name="rlimit" type="xs:int"/>

     <xs:element name="rpngx" type="xs:float"/>

  </xs:sequence>
</xs:complexType>
```

Logical Elements

TRESYS Deep.

DFDL schema

```
<xs:annotation>
  <xs:appinfo source="http://www.ogf.org/dfdl/v1.0">
     <dfdl:format representation="text"
             textNumberRep="standard" encoding="ascii"
             lengthKind="delimited" .../>
   </xs:appinfo>
</xs:annotation>

<xs:complexType name="rValues">
  <xs:sequence dfdl:separator=";" ... >
    <xs:element name="rLimit" type="xs:int"
                dfdl:initiator="rLimit=" ... />
     <xs:element name="rpngx" type="xs:float"
                dfdl:initiator="rpngx=" ... />
  </xs:sequence>
</xs:complexType>
```

DFDL properties

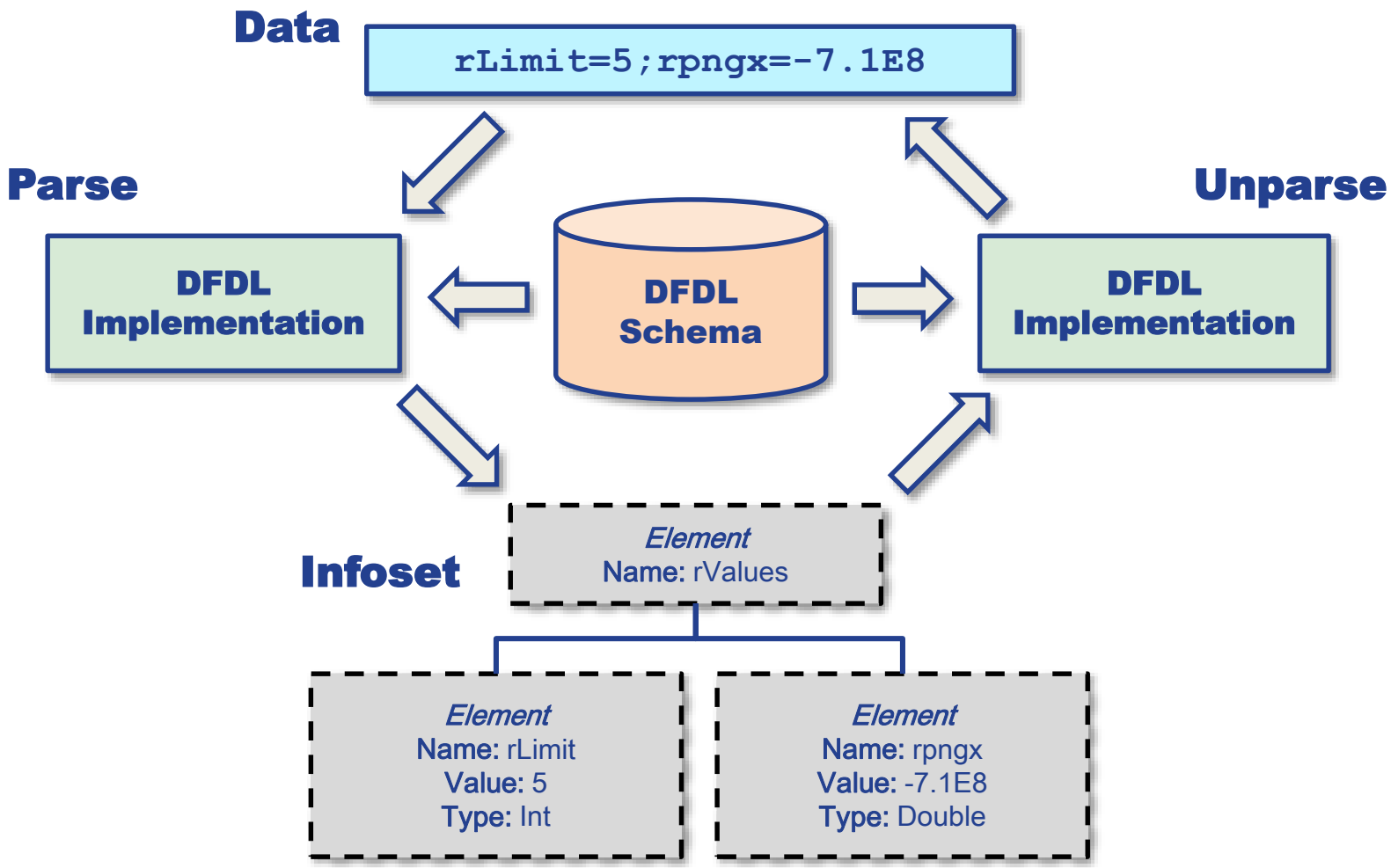
;

rLimit=5

rpngx=-7.1E8

TRESYS Deep.

DFDL Lifecycle



TRESYS Deep.

DFDL Use Cases

- Deep Content Inspection/Sanitization
  - CDS, Network Guards
  - Data Filtering
  - Rip-and-Rebuild Firewalls
- Data-Directed Routing
- Data Intake
  - Data Warehouse or Analytical Data Store
- Interoperability / Data Translation
- Conversion of Legacy Data to Modern Formats
- Open Data / Data Export
  - Recasting data as XML for export/publishing

CSV (ASCII text delimited)

PCAP (variable length binary)

MIL-STD-2045 (dense bit-packed binary, LSBF)

EXAMPLES

TRESYS Deep.

Comma-Separated Values

EXAMPLE: CSV

TRESYS Deep.

Example - CSV

- Textual Format (ASCII)
- Comma-Separated Values
- Tabular data
- Rows separated by newlines
- Fields separated by commas

```
Smith,Charles,36,6.2
Williams,Mary,51,5.5
```

Example - CSV

```
<xs:schema>
  <xs:annotation>
    <xs:appinfo source="http://www.ogf.org/dfdl/">
      <dfdl:format ref="defaults"
                   representation="text"
                   encoding="ASCII"
                   lengthKind="delimited"
                   escapeKind="escapeBlock"
                   escapeBlockStart="&quot;"
                   escapeBlockEnd="&quot;"
                   escapeEscapeCharacter="&quot;"
                   occursCountKind="parsed" />
    </xs:appinfo>
  <xs:annotation>


<xs:schema>
```

TRESYS Deep.

Example - CSV

```
<xs:schema>
  <xs:annotation>
    <xs:appinfo source="http://www.ogf.org/dfdl/">
      <dfdl:format ref="defaults"
                   representation="text"
                   encoding="ASCII"
                   lengthKind="delimited"
                   escapeKind="escapeBlock"
                   escapeBlockStart="&quot;"
                   escapeBlockEnd="&quot;"
                   escapeEscapeCharacter="&quot;"
                   occursCountKind="parsed" />
    </xs:appinfo>
  <xs:annotation>


<xs:schema>
```

TRESYS Deep.

Example - CSV

```
<xs:element name="csv">
  <xs:complexType>
    <xs:sequence dfdl:separator="%NL;"
                 dfdl:separatorPosition="postfix">
      <xs:element name="record" maxOccurs="unbounded">
        <xs:complexType>
          <xs:sequence dfdl:separator=","
                       dfdl:separatorPosition="infix">
            <xs:element name="item" type="xs:string"
                        maxOccurs="unbounded"/>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

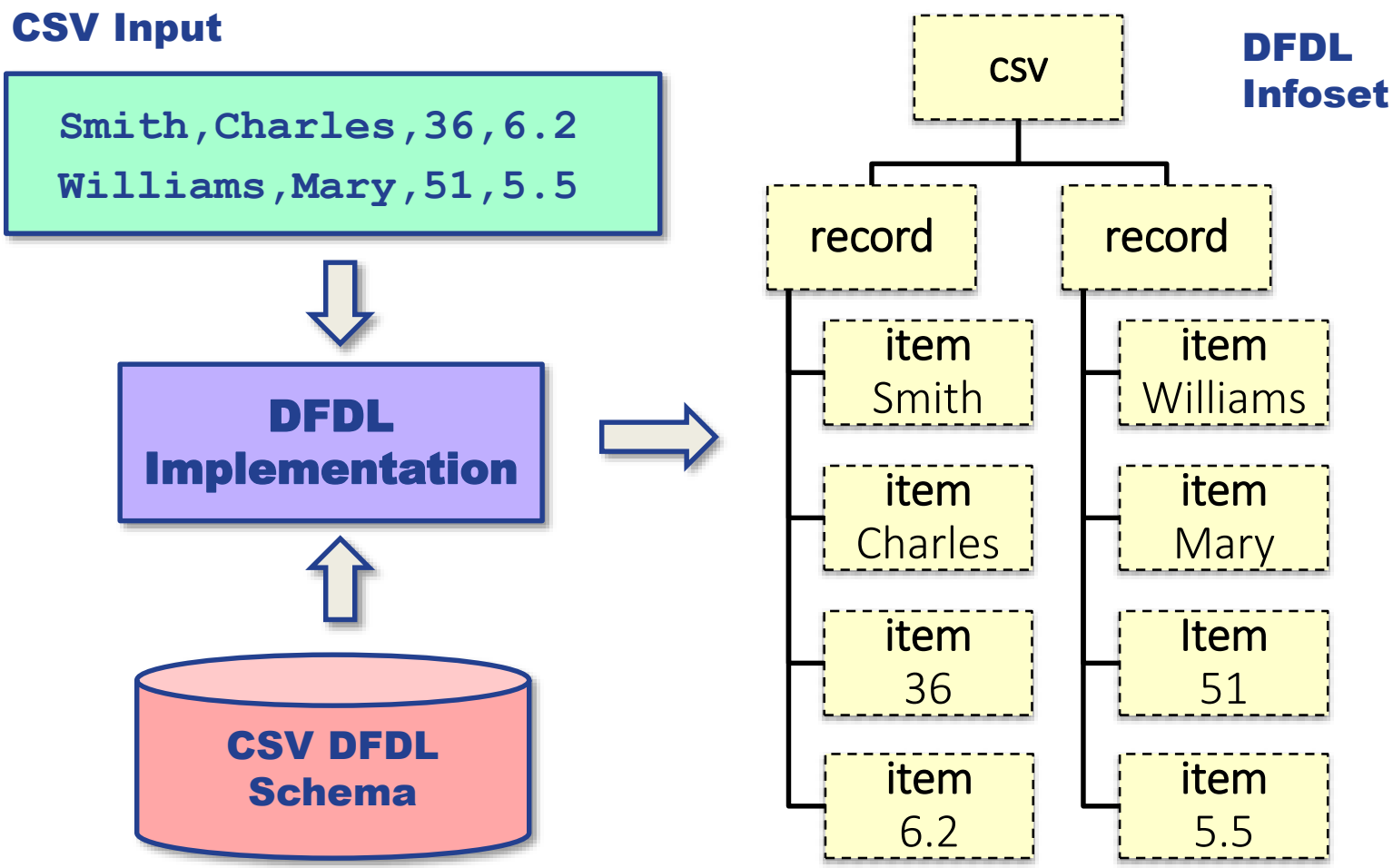
TRESYS Deep.

Example - CSV

```
<xs:element name="csv">
  <xs:complexType>
    <xs:sequence dfdl:separator="%NL;"
                 dfdl:separatorPosition="postfix">
      <xs:element name="record" maxOccurs="unbounded">
        <xs:complexType>
          <xs:sequence dfdl:separator=","
                       dfdl:separatorPosition="infix">
            <xs:element name="item" type="xs:string"
                        maxOccurs="unbounded"/>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

TRESYS Deep.

Example - CSV


CSV Input
Smith,Charles,36,6.2
Williams,Mary,51,5.5
DFDL Implementation
CSV DFDL Schema
csv
DFDL Infoset
record
record
item Smith
item Charles
item 36
item 6.2
item Williams
item Mary
Item 51
item 5.5

Packet Capture File Format

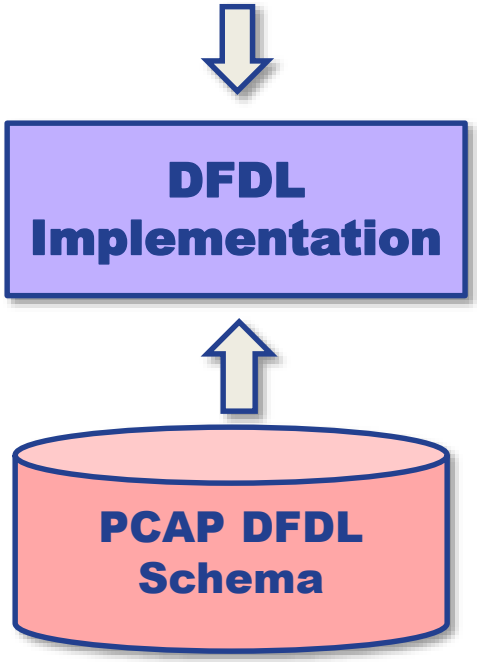
EXAMPLE: PCAP

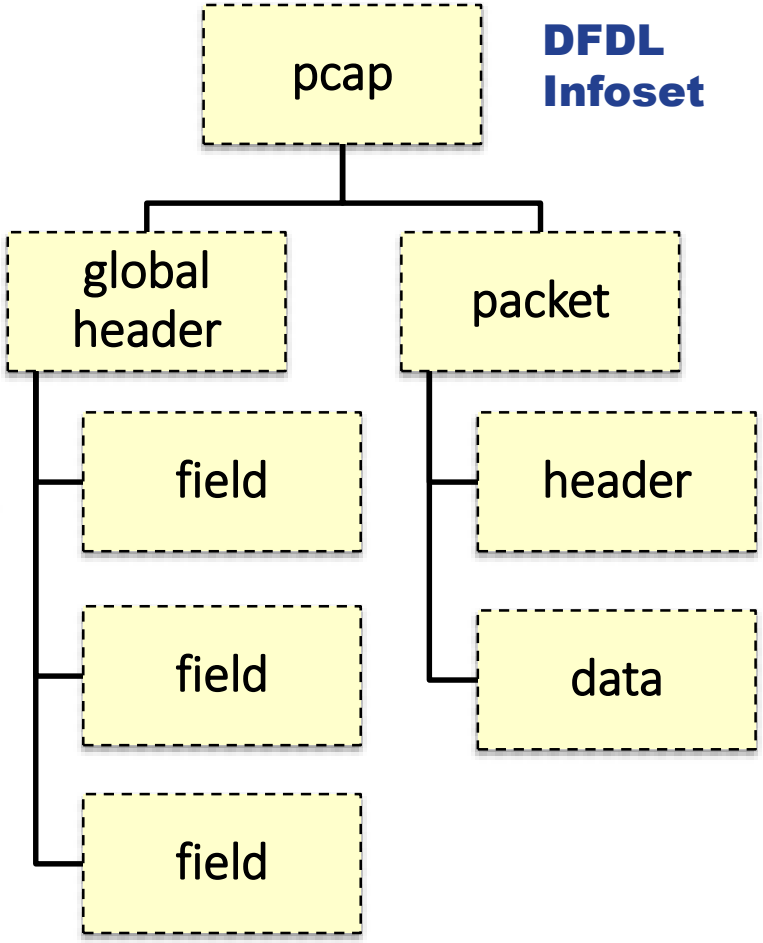
TRESYS Deep.

Example - PCAP

PCAP Data

```
c3d4 a1b2 0002 0004 0000 0000 0000 0000
9000 0001 0001 0000 eb1d 42d2 0000 0000
0040 0000 0040 0000 8001 00c2 0e00 0400
1f96 26a7 cc88 0702 0004 9604 a71f 0426
0504 2f31 0633 0002 0678 0002 0601 0002
0602 0002 0003 ff00 ffff ffff ffff ffff
ffff ffff ffff bbaa
```

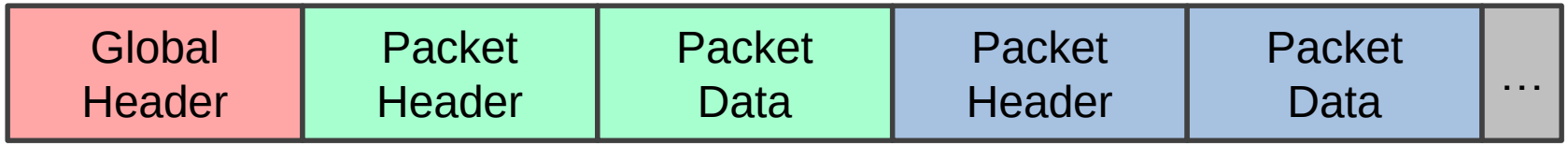




TRESYS
Deep.

Example - PCAP

- Network Packet Capture
- Binary Format
- Global Header
- Packet Header/Data
- Variable Endianness and Data Length



Example - PCAP

```
<xs:schema>
  <xs:annotation>
    <xs:appinfo source="http://www.ogf.org/dfdl/">
      <dfdl:format ref="defaults"
                   representation="binary"
                   alignment="8"
                   alignmentUnits="bits"
                   lengthUnits="bits"
                   binaryNumberRep="binary"
                   lengthKind="implicit"
                   byteOrder="{ $byte_order }" />
      <dfdl:defineVariable name="byte_order" type="xs:string"/>
    </xs:appinfo>
  </xs:annotation>

  <xs:simpleType name="uint32" dfdl:lengthKind="explicit"
                               dfdl:length="32">
    <xs:restriction base="xs:unsignedInt"/>
  </xs:simpleType>
</xs:schema>
```

TRESYS Deep.

Example - PCAP

```
<xs:schema>
  <xs:annotation>
    <xs:appinfo source="http://www.ogf.org/dfdl/">
      <dfdl:format ref="defaults"
                   representation="binary"
                   alignment="8"
                   alignmentUnits="bits"
                   lengthUnits="bits"
                   binaryNumberRep="binary"
                   lengthKind="implicit"
                   byteOrder="{ $byte_order }" />
      <dfdl:defineVariable name="byte_order" type="xs:string"/>
    </xs:appinfo>
  </xs:annotation>

  <xs:simpleType name="uint32" dfdl:lengthKind="explicit"
                               dfdl:length="32">
    <xs:restriction base="xs:unsignedInt"/>
  </xs:simpleType>
</xs:schema>
```

TRESYS Deep.

Example - PCAP

```
<xs:schema>
  <xs:annotation>
    <xs:appinfo source="http://www.ogf.org/dfdl/">
      <dfdl:format ref="defaults"
                   representation="binary"
                   alignment="8"
                   alignmentUnits="bits"
                   lengthUnits="bits"
                   binaryNumberRep="binary"
                   lengthKind="implicit"
                   byteOrder="{ $byte_order }" />
      <dfdl:defineVariable name="byte_order" type="xs:string"/>
    </xs:appinfo>
  </xs:annotation>

  <xs:simpleType name="uint32" dfdl:lengthKind="explicit"
                               dfdl:length="32">
    <xs:restriction base="xs:unsignedInt"/>
  </xs:simpleType>
</xs:schema>
```

TRESYS Deep.

Example - PCAP

```
<xs:element name="magic_number" type="ex:uint32"
            dfdl:byteOrder="bigEndian">
  <xs:annotation>
    <xs:appinfo source="http://www.ogf.org/dfdl/">
      <dfdl:setVariable ref="byte_order"><![CDATA[{
        if (. eq 2712847316) then 'bigEndian'
        else if (. eq 3569595041) then 'littleEndian'
        else daf:error() }]]>
      </dfdl:setVariable>
      <dfdl:property name="outputValueCalc"><![CDATA[{
           if ($dfdl:byteOrder eq 'bigEndian')
           then xs:hexBinary('A1B2C3D4')
           else xs:hexBinary('D4C3B2A1')
           }]]></dfdl:property>

    </xs:appinfo>
  </xs:annotation>
</xs:element>
```

TRESYS Deep.

Example - PCAP

```
<xs:element name="magic_number" type="ex:uint32"
            dfdl:byteOrder="bigEndian">
  <xs:annotation>
    <xs:appinfo source="http://www.ogf.org/dfdl/">
      <dfdl:setVariable ref="byte_order"><![CDATA[{
        if (. eq 2712847316) then 'bigEndian'
        else if (. eq 3569595041) then 'littleEndian'
        else daf:error() }]]>
      </dfdl:setVariable>
      <dfdl:property name="outputValueCalc"><![CDATA[{
           if ($dfdl:byteOrder eq 'bigEndian')
           then xs:hexBinary('A1B2C3D4')
           else xs:hexBinary('D4C3B2A1')
           }]]></dfdl:property>

    </xs:appinfo>
  </xs:annotation>
</xs:element>
```

TRESYS Deep.

Example - PCAP

```
<xs:element name="PacketHeader">
   <xs:complexType>
      <xs:sequence>
         <xs:element name="Seconds" type="pcap:uint32"/>
         <xs:element name="USeconds" type="pcap:uint32"/>
         <xs:element name="InclLen" type="pcap:uint32"
            ...
         />
         <xs:element name="OrigLen" type="pcap:uint32"
            ...
          />
      </xs:sequence>
   </xs:complexType>
</xs:element>
...
<xs:element ref="pcap:LinkLayer"
   dfdl:lengthUnits="bytes" dfdl:lengthKind="explicit"
   dfdl:length="{ ../PacketHeader/InclLen }"/>
```

TRESYS Deep.

Example - PCAP

```
<xs:element name="PacketHeader">
   <xs:complexType>
      <xs:sequence>
         <xs:element name="Seconds" type="pcap:uint32"/>
         <xs:element name="USeconds" type="pcap:uint32"/>
         <xs:element name="InclLen" type="pcap:uint32"
            ...
         />
         <xs:element name="OrigLen" type="pcap:uint32"
            ...
          />
      </xs:sequence>
   </xs:complexType>
</xs:element>
...
<xs:element ref="pcap:LinkLayer"
   dfdl:lengthUnits="bytes" dfdl:lengthKind="explicit"
   dfdl:length="{ ../PacketHeader/InclLen }"/>
```

TRESYS Deep.

Example - PCAP

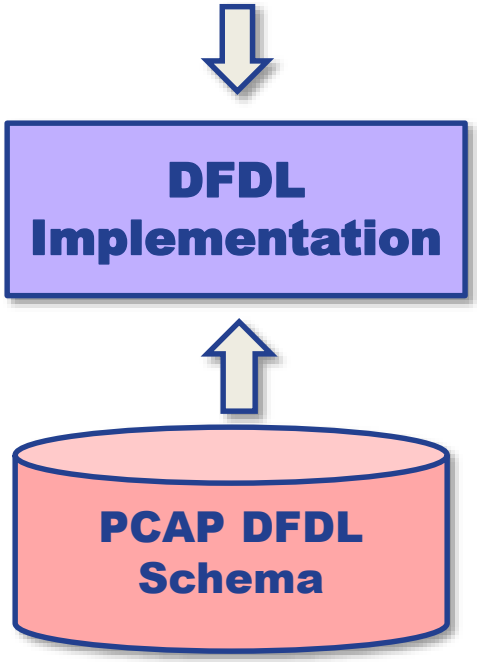
```
<xs:element name="PacketHeader">
   <xs:complexType>
      <xs:sequence>
         <xs:element name="Seconds" type="pcap:uint32"/>
         <xs:element name="USeconds" type="pcap:uint32"/>
         <xs:element name="InclLen" type="pcap:uint32"
           dfdl:outputValueCalc="{
            if (dfdl:valueLength(
                      ../../pcap:LinkLayer/pcap:Ethernet,
                      'bytes') le 60) then 60
            else
               dfdl:valueLength(
                      ../../pcap:LinkLayer/pcap:Ethernet,
                      'bytes') }"
         />
         <xs:element name="OrigLen" type="pcap:uint32"
            dfdl:outputValueCalc="{ ../InclLen }"/>
      </xs:sequence>
   </xs:complexType>
</xs:element>
...
<xs:element ref="pcap:LinkLayer"
   dfdl:lengthUnits="bytes" dfdl:lengthKind="explicit"
   dfdl:length="{ ../PacketHeader/InclLen }"/>
```

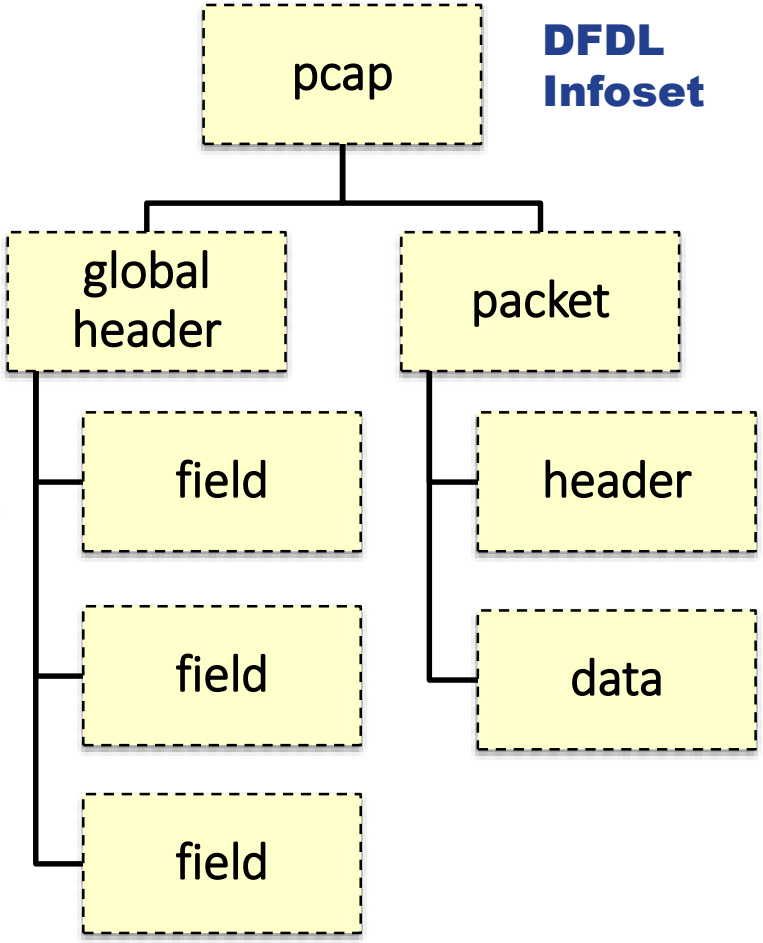
TRESYS Deep.

Example - PCAP

PCAP Data

```
c3d4 a1b2 0002 0004 0000 0000 0000 0000
9000 0001 0001 0000 eb1d 42d2 0000 0000
0040 0000 0040 0000 8001 00c2 0e00 0400
1f96 26a7 cc88 0702 0004 9604 a71f 0426
0504 2f31 0633 0002 0678 0002 0601 0002
0602 0002 0003 ff00 ffff ffff ffff ffff
ffff ffff ffff bbaa
```





TRESYS
Deep.

A dense bit-packed binary data format similar to many MIL-STD formats, but publicly available.

Uses Least-Significant-Bit-First bit order – typical of MIL formats, different from most non-MIL formats.

EXAMPLE: MIL-STD-2045

Example - MIL-STD-2045

- Densely packed binary with unique bit order

```
dfdl:representation="binary"
dfdl:alignment="1"
dfdl:alignmentUnits="bits"
dfdl:lengthUnits="bits"
dfdl:lengthKind="explicit"
dfdl:length
dfdl:byteOrder="littleEndian"
dfdl:bitOrder="leastSignificantBitFirst"
```

Example - MIL-STD-2045

- 7-bit strings embedded in binary

```
dfdl:encoding="X-DFDL-US-ASCII-7BIT-PACKED"
```

- 0x7F (DEL) terminated, unless max-length met

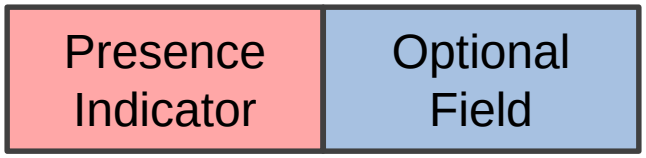
```
<xs:element name="str50" type="xs:string"
            dfdl:lengthPattern="[^\x7F]{0,49}(?=\x7F)|.{50}" />

<xs:sequence dfdl:terminator="{
  if (fn:string-length(./str50) eq 50)
  then '%ES;'
  else '%DEL;'
}" />
```

TRESYS Deep.

Example - MIL-STD-2045

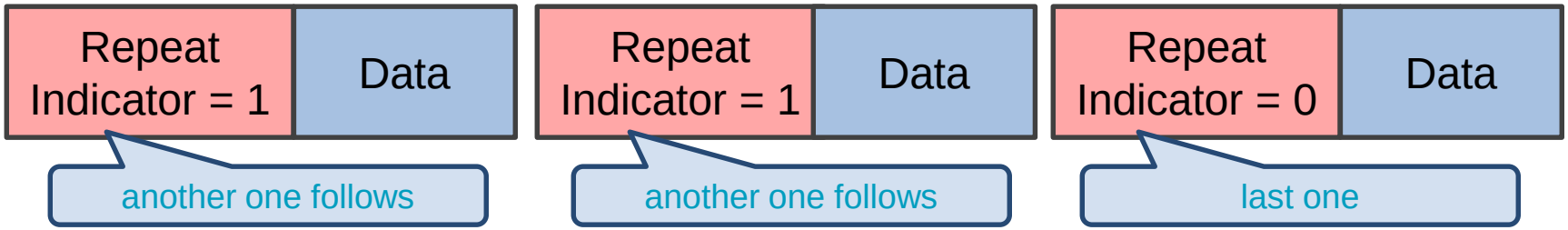
- Presence indicators




```
<dfdl:discriminator test="{ . eq 1 }" />
```

Example - MIL-STD-2045

- Repeat indicators




```
dfdl:occursCountKind="implicit"
minOccurs="1"
maxOccurs="unbounded"

<dfdl:discriminator test="{
  if (dfdl:occursIndex() eq 1)
  then fn:true()
  else ../fields[dfdl:occursIndex() - 1]/repeat_indicator eq 1
}" />

 <xs:element name="GRI" type="tns:repeatIndicator"
   dfdl:outputValueCalc="{
     if (dfdl:occursIndex() lt fn:count(..))
     then 1 else 0
 }" />
```

More DFDL Properties

```
dfdl:assert
dfdl:discriminator
dfdl:newVariableInstance
dfdl:setVariable
dfdl:defineEscapeSchema
dfdl:defineVariable
dfdl:ignoreCase
dfdl:encodingErrorPolicy
dfdl:alignment
dfdl:alignmentUnits
dfdl:fillByte
dfdl:leadingSkip
dfdl:trailingSkip
dfdl:emptyValueDelimiterPolicy
dfdl:outputNewLine
dfdl:prefixIncludesPrefixLength
dfdl:prefixLengthType
dfdl:lengthPattern
dfdl:textPadKind
dfdl:textTrimKind
dfdl:textOutputMinLength
dfdl:escapeSchemeRef
dfdl:escapeKind
dfdl:escapeCharacter
dfdl:escapeBlockStart
dfdl:escapeBlockEnd
dfdl:escapeEscapeCharacter
dfdl:extraEscapedCharacters
dfdl:generateEscapeBlock
dfdl:textBidi
dfdl:textStringJustification
dfdl:textStringPadCharacter
dfdl:textNumberJusitification
dfdl:textStandardNaNRep
dfdl:textStandardZeroRep
dfdl:textStandardBase
dfdl:binaryNumberRep
dfdl:binaryPackedSignCodes
dfdl:binaryFloatRep
dfdl:calendarPattern
dfdl:binaryCalendarRep
dfdl:nilValueDelimiterPolicy
dfdl:initiatedContent
dfdl:separatorPosition
dfdl:separatorSuppressionPolicy
dfdl:floating
dfdl:hiddenGroupRef
dfdl:initiatedContent
dfdl:occursCountKind
dfdl:occursStopValue
dfdl:inputValueCalc
dfdl:outputValueCalc
```

TRESYS Deep.

```
Hello world!
```

```
<helloWorld>
  <word>Hello</word>
  <word>world!</word>
</helloWorld>
```

Yes, it is 'Hello world!'

EXAMPLE: JAVA API

TRESYS Deep.

Daffodil Java API – Hello World

```
public class HelloWorld {
  public static void main(String[] args) throws IOException {
     ... // let's fill this in
  }
}
```

```
 <!- helloWorld.dfdl.xsd -->
  …
<dfdl:format ref="daffodilTest1
   representation="text" encoding="utf-8"
   lengthKind="delimited"/>
  …
<xs:element name="helloWorld">
  <xs:complexType><xs:sequence>

        <xs:element name="word" type="xs:string"
                maxOccurs="unbounded"
                dfdl:lengthKind="delimited"
                dfdl:terminator="%WSP+;"
                dfdl:occursCountKind="parsed"/>

    </xs:sequence></xs:complexType>
</xs:element>
```

TRESYS Deep.

Daffodil Java API – Hello World

```
//
// First compile the DFDL Schema
// Creates a Daffodil ProcessorFactory object
//
Compiler c = Daffodil.compiler();
c.setValidateDFDLSchemas(true);

File schemaFile = new File("helloWorld.dfdl.xsd");
ProcessorFactory pf = c.compileFile(schemaFile);

// list any compile-time diagnostics - warnings and errors
List<Diagnostic> diags = pf.getDiagnostics();
for (Diagnostic d : diags) {
  System.err.println(d.getSomeMessage());
}
if (pf.isError()) { System.exit(1); }
```

TRESYS Deep.

Daffodil Java API – Hello World

```
//
// Use ProcessorFactory to create data processors
// As many as you want. They can be run (to parse/unparsed) on
// different threads.
//
DataProcessor dp = pf.onPath("/");

ReadableByteChannel rbc =
    Channels.newChannel(new FileInputStream("helloWorld.dat"));

ParseResult res = dp.parse(rbc);

boolean err = res.isError();
if (err) { … list diagnostics and exit(2) … }

org.jdom2.Document doc = res.result(); // JDOM2 XML Infoset
```

TRESYS Deep.

Daffodil Java API – Hello World

```
//
// Pretty print the XML infoset.
//
XMLOutputter xo = new XMLOutputter(Format.getPrettyFormat());
xo.output(doc, System.out);

// or transform/inspect/etc.
```

TRESYS Deep.

Daffodil Java API – Hello World

```
//
// unparse
//

WritableByteChannel wbc =
  Channels.newChannel(new FileOutputStream(…));

UnparseResult res2 = dp.unparse(wbc, doc);

if (res2.isError()) { … same as before … }

// File contains unparsed data
```

TRESYS Deep.

DFDL Schemas

* = in development
** = not yet published

Public (github)	MIL-STD-2045 PCAP NITF* PNG** JPEG** NACHA vCard	EDIFACT IBM4690-TLOG ISO8583 planned: GeoTIFF, Exif, TIFF EP, DNG, WMF, EMF, GIF, BMP, ...
FOUO (forge.mil)	MIL-STD-6017/VMF(PL only) MIL-STD-6040/USMTF (ATO only) MIL-STD-6016/Link16/TDL-J (J9, J3.2, J2.0, J3.5,...) * A-GNOSC REMEDY ARMY DRRS USCG UCOP CEF-R1965 GMTIF (STANAG 4607)*	
Commercial License $$$	SWIFT-MT (IBM) HIPAA-5010 (IBM) HL7-2.7 (IBM)	

- Daffodil – Open Source
- IBM DFDL
- DFDL4S – from ESA

DFDL IMPLEMENTATIONS

TRESYS Deep.

Daffodil

- Open Source
  - University of Illinois/NCSA Open Source License
    - A Permissive/Commercial Friendly License – embed how you wish, into anything.
- Hosted by University of Illinois/NCSA
- Runs on Java Virtual Machine
- Started as research project by UofI ~2009
- Primarily developed by Tresys Technology ~2012
  - Funded by the USG
- Subset of full DFDL features – oriented toward our customers' needs
  - Packed decimal – no. Bit order – yes. Computed elements = yes.
- Active development

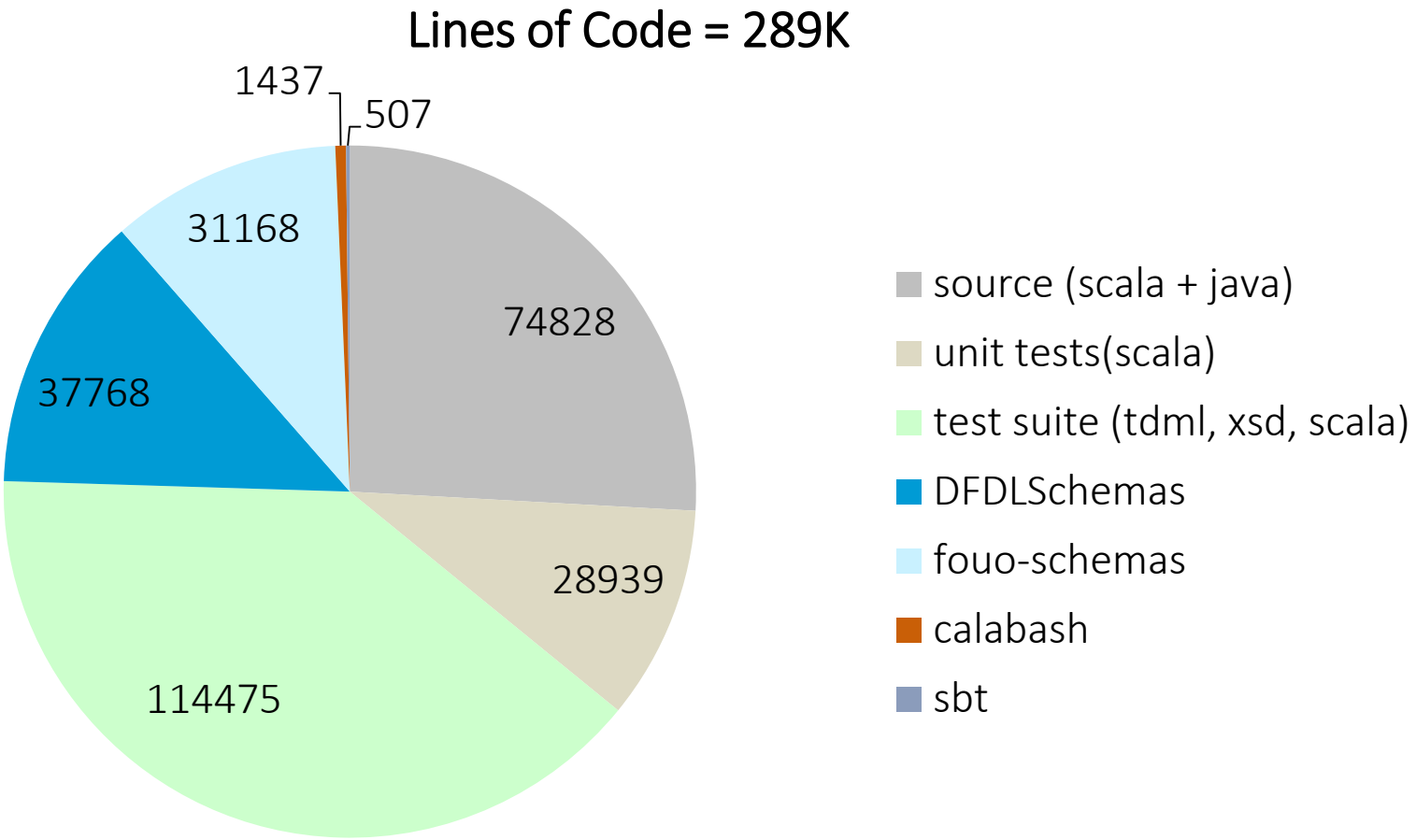
- https://opensource.ncsa.illinois.edu/confluence/display/DFDL/

TRESYS Deep.

Daffodil

- Command Line Interface
- Interactive CLI Debugger and Trace
- Java & Scala API with documentation
- Calabash Extension for XProc pipeline

Daffodil Code Base



Data as of 2017-01-25

Daffodil Integrations

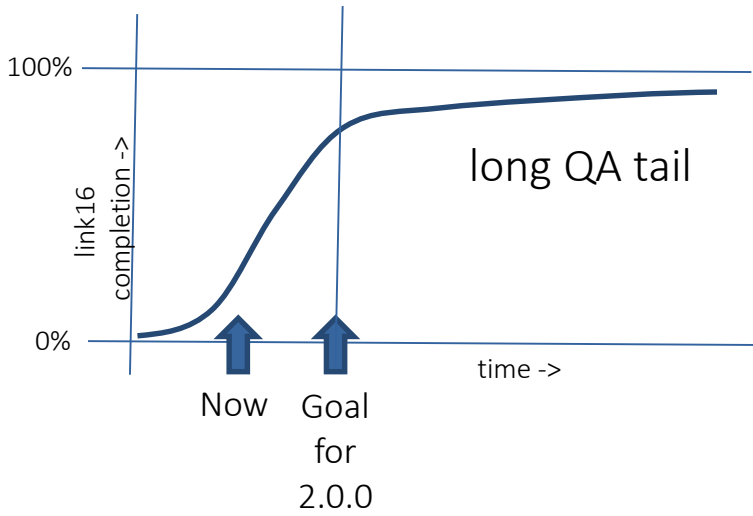
- ✓ XProc - Calabash
- ✓ Apache Spark,
- ✓ Apache NiFi
- ✓ Apache Storm
  - demonstrated by Quark Security. Published ???
- ✓ Tresys CDS products/services
  - details are FOUO

- Potential
  - Apache Tika
  - Other data-handling frameworks - Storm, Flink, Hadoop,....

TRESYS Deep.

Next major release - Daffodil 2.0.0

- Performance improvements
  - Unparse speed
  - Conversion speed to/from XML
- Functionality/fixes needed for:
  - Link16
  - NITF with embedded JPEGs
  - TIFF
- ✓ Apache NiFi Integration



Daffodil Future Development

- DFDL Schemas for important formats
  - MIL Standards – TDL/J, VMF, USMTF
- Cross-validation/test with IBM DFDL
  - Packed/zoned decimal and other IBM-legacy formats
- Tutorials on writing/debugging DFDL schemas
- Improved Trace and Debug
- Streaming behavior for parsing, unparsing
- Separate Compilation for large schemas
- Integrate into important frameworks: Spark, Storm, NiFi, etc.
- JSON support
- Extensions: recursion, layering, extensibility, document formats, BLOB/CLOB

IBM© DFDL

- Designed to be embeddable or standalone
- Includes parser and unparser
- Available in C and Java
- Emits SAX-style events/Accepts SAX-style events
- Includes Eclipse based tooling
  - Graphical DFDL schema editor/test environment
- Subset of full DFDL features – oriented toward IBM's customers' needs
  - Packed decimal – yes. Bit order – no. Computed elements – no.
- Part of IBM WebSphere Product Family
  - www.ibm.com/developerworks/library/se-dfdl/index.html

DFDL4S from ESA

- DFDL4S = DFDL for Space
- Binary-data-only subset of DFDL
- Created by European Space Agency (ESA) for satellite data descriptions
- In use now.
- Evolving to be a fully compatible DFDL subset.
  - DFDL standard specifically allows for specialized subsets like this to be accepted as conforming.
- More info at
  - http://eop-cfi.esa.int/index.php/applications/dfdl4s

Tresys DFDL Products and Services

We would like to know how to make DFDL and Daffodil solve your data challenges!

Example data challenges

- Data Intake to analysis systems
- Data Publishing – e.g., USG Open Data requirements
- Data Security – CDS/ rip, inspect/transform, rebuild to specification

Examples of services Tresys can provide:

- Create (or help you create) & Test DFDL Schemas for
  - Your specific in-house data formats
  - Industry/Military standard data formats
- Enhance Daffodil to meet your needs
  - Customize, add features, APIs, ….
- Support your team embedding Daffodil into products

TRESYS Deep.

QUESTIONS?

DFDL Specification:

http://ogf.org/dfdl

DFDL Schemas:

https://github.com/DFDLSchemas

Daffodil Wiki:

https://opensource.ncsa.illinois.edu/confluence/display/DFDL/

TRESYS Deep.

END

TRESYS
Deep.