DFDL Enhancements

Bradd Kadlecik
z/TPF Development

Background

- DFDL (Data Format Description Language) is a universal, shareable, non-prescriptive description for general text and binary data formats.

- DFDL can be used to transform data between native, proprietary formats and standardized formats such as XML, JSON, BSON, and CSV.

- DFDL is integrated in z/TPF into business events, support for MongoDb, and REST.

Agenda

- **What's New:**
  - Calculated fields
  - Hidden groups
  - User-exit functions
  - Pointers
- **What's Next:**
  - Default field exclusion
  - Flatten/unflatten

Calculated fields

Problem

- Variable length elements and variable size arrays require length and size elements to be part of the DFDL description.

- The length and size elements are not transmitted as separate elements in standardized formats but rather are implicitly understood.

Pain Points

- Passing the length of elements and size of arrays as separate elements can cause corruption by inaccurate values being passed, not to mention an unconventional interface.

- Excluding the length and size elements causes DFDL to not be used (or used piecemeal) causing a much higher development cost.

Value Statement

- DFDL calculated fields (PJ45427) allows standardized formats involving variable length elements and variable size arrays to be defined in DFDL in such a way as to not require the passing in of length and size elements.

The Details

The DFDL annotation **dfdl:outputValueCalc** provides:

- Ability to set the length of a variable size string based on the length of the string in the document using the **dfdl:valueLength**(<XPath>) function.
- Ability to set the size of a variable size array based on the number of occurrences in the document using the **fn:count**(<XPath>) function.
- Ability to set existence information for optional fields using the **fn:exists**(<XPath>) function.

Note: Only used during DFDL serialization!

Example – variable length field

```
<xs:element name="addrLen" type="xs:unsignedShort"
        dfdl:lengthKind="explicit" dfdl:length="2"
        dfdl:lengthUnits="bytes" default="0"
        dfdl:outputValueCalc="{dfdl:valueLength(../Address,'bytes')}" />

<xs:element name="Address" type="xs:string"
        dfdl:lengthKind="explicit" dfdl:length="{../addrLen}"
        dfdl:lengthUnits="bytes" />
```

Example – variable length field

JSON format:
“Address”:”100 Main St”

dfdl:valueLength(../Address,’bytes’) = 11

Binary Format:
11 “100 Main St”
000B F1F0F040D481899540E2A3

Example – variable size array

```
<xs:schema xmlns:fn="http://www.w3.org/2005/xpath-functions" >

<xs:element name="arrayCount" type="xs:unsignedShort"
        dfdl:lengthKind="explicit" dfdl:length="2"
        dfdl:lengthUnits="bytes“ default="0"
        dfdl:outputValueCalc="{fn:count(../Numbers)}"
<xs:element name=“Numbers" type="xs:int"
        dfdl:lengthKind="explicit" dfdl:length=“4"
        dfdl:lengthUnits="bytes" minOccurs="0"
        maxOccurs="unbounded" dfdl:occursCountKind="expression"
        dfdl:occursCount="{../arrayCount}" />
```

Example – variable size array

JSON format:
“Numbers”:[10,20,30,40]

fn:count(../Numbers) = 4

Binary Format:

4	10	20	30	40
0004	0000000A	00000014	0000001E	00000028

Example – optional field

```
<xs:schema xmlns:fn="http://www.w3.org/2005/xpath-functions" >

<xs:element name=“hasAddr" type="xs:boolean"
        dfdl:lengthKind="explicit" dfdl:length=“1"
        dfdl:lengthUnits="bits" dfdl:alignmentUnits="bits" default="0"
        dfdl:leadingSkip="2" dfdl:trailingSkip="5"
        dfdl:outputValueCalc="{fn:exists(../Address)}" />
<xs:element name="Address" type="xs:string"
        dfdl:lengthKind="explicit" dfdl:length="20"
        dfdl:lengthUnits="bytes" nillable="true"
        dfdl:useNilForDefault="yes" />
```

Hidden groups

Problem

- Some fields have no value in being transmitted (such as lengths of variable length elements and sizes of variable size arrays).

- Need to restrict whether or not certain fields are transmitted based on where the data is going (different views of the same data).

Pain Points

- A complicated update to DFDL needs to be made in order to keep certain elements from being transmitted. This approach is not accepted on all DFDL parsers when the hidden element is referenced through a DFDL expression.

- Creating multiple DFDLs for the same data for the purpose of having different views can be very difficult to keep in synch as changes are made.

Value Statement

- DFDL hidden groups (PJ45427) provides a standardized way to define multiple views of the data by restricting certain fields from being transmitted.

The Details

The DFDL annotation **dfdl:hiddenGroupRef** provides:

- Ability to restrict a field or set of fields from being externalized in standardized formats.
- Ability to conditionally restrict a field or set of fields from being externalized through DFDL variables, allowing for different views based on the value(s) of either a **DFDL variable or name-value pair**.

Example – hidden length field

```
<xs:group name="hiddenLen">
   <xs:sequence>
      <xs:element name="addrLen" type="xs:unsignedShort"
           dfdl:lengthKind="explicit" dfdl:length="2" dfdl:lengthUnits="bytes"
           dfdl:outputValueCalc="{dfdl:valueLength(../Address,'bytes')}" />
   </xs:sequence>
</xs:group>

<xs:sequence dfdl:hiddenGroupRef="hiddenLen"/>
<xs:element name="Address" type="xs:string"
           dfdl:lengthKind="explicit" dfdl:length="{../addrLen}"
           dfdl:lengthUnits="bytes" />
```

Example – alternate views

```
{
  “name”:”John Smith”,
  “sex”:”M”,
  “age”:40,
  “eyes”:”brown”
}
```

```
{
  “name”:”John Smith”,
  “sex”:”M”,
  “eyes”:”brown”
}
```

Example – alternate views (contd)

```
<!-- Define an external variable to control the view -->
<xs:annotation>
   <xs:appinfo source="http://www.ogf.org/dfdl/">
      <dfdl:defineVariable name="allowPII" type="xs:boolean" external="true" />
   </xs:appinfo>
</xs:annotation>

<xs:group name="PII_Info">
   <xs:sequence>
      <xs:element name="age" type="xs:unsignedByte"
           dfdl:lengthKind="explicit" dfdl:length="1"
           dfdl:lengthUnits="bytes" default="0" />
    </xs:sequence>
</xs:group>
```

Example – alternate views (contd)

```
<xs:choice>
   <xs:group ref="PII_Info">
      <xs:annotation>
         <xs:appinfo source="http://www.ogf.org/dfdl/">
            <dfdl:discriminator>{$allowPII}</dfdl:discriminator>
         </xs:appinfo>
      </xs:annotation>
   </xs:group>
   <xs:sequence dfdl:hiddenGroupRef="PII_Info" />
</xs:choice>
```

Example – alternate views (contd)

```
try {
  tpf_dfdl_initialize_handle(&dh, DFDL_FILE, DFDL_ROOT, 0);
  tpf_dfdl_setData(dh, &buf, sizeof(buf));
  val.dataType = DFDL_TYPE_BOOLEAN;
  val.value.v_ulong = 1;
  tpf_dfdl_setVariable(dh, “allowPII", &val, NULL);
  JSONdoc = tpf_dfdl_buildDoc(dh, &docLen, TPF_DFDL_JSON, 0);
}
```

Example – name-value pair

```
<xs:schema xmlns:nv="http://www.ibm.com/xmlns/prod/ztpf/name-value" >

<xs:choice>
   <xs:group ref="PII_Info">
      <xs:annotation>
         <xs:appinfo source="http://www.ogf.org/dfdl/">
            <dfdl:discriminator>{$nv:allowPII}</dfdl:discriminator>
         </xs:appinfo>
      </xs:annotation>
   </xs:group>
   <xs:sequence dfdl:hiddenGroupRef="PII_Info" />
</xs:choice>
```

User exit functions

Problem

- Some fields in a JSON or XML document might not come from a single data structure but instead be associated with the ECB or time of transaction.

- Some fields require special formatting to convert between the native format in the data structure and the string format in the document.

Pain Points

- Copying data to new structures that contain all the data needed by a document or modifying infonodes after DFDL creation can be costly and error prone.

- Special formatting performed by one group using DFDL is not reusable by other groups without good design and coordination.

Value Statement

- DFDL calculated fields (PJ45427) provides a standardized way to add element values to a document that doesn’t come from the data.

- DFDL user-exit functions (PJ45427) provides the ability to create and maintain conversion/formatting functions shareable with all DFDL schemas.

The Details

The DFDL annotation **dfdl:inputValueCalc** provides:

- Ability to set the value of an element in a document that's not defined in the data structure by using a **DFDL variable**.
- Ability to peform special formatting for an element by using a **ufn:<user function>(XPath)** function.

Example – data augmentation

Binary:

```
typedef struct {
   unsigned int iob;
   unsigned int frame;
   unsigned int common;
   unsigned int swb;
   unsigned int xwb;
   unsigned int ecb;
   unsigned int frm1mb;
} resource_availablity;
```

JSON:

```
{
   “resource_availablity”:{
      “time”:”2019-03-01T09:30:01”,
      “iob”:8192,
      “frame”:6192,
      “common”:397,
      “swb”:3890,
      “xwb”:3946,
      “ecb”:558,
      “frm1mb”:283
   }
}
```

Example – data augmentation

```
<!-- Define an external variable to import a string value -->
<xs:annotation>
   <xs:appinfo source="http://www.ogf.org/dfdl/">
      <dfdl:defineVariable name="dateTime" type="xs:string" external="true" />
   </xs:appinfo>
</xs:annotation>

<xs:complexType name="resource_availablity">
   <xs:sequence>
      <xs:element name="time" type="xs:string" dfdl:inputValueCalc="{$ns0:dateTime}" />
      <xs:element name="iob" type="xs:unsignedInt" dfdl:lengthKind="explicit"
           dfdl:length="4" dfdl:lengthUnits="bytes" default="0" />
      ...
```

Example – data augmentation

```
try {
  tpf_dfdl_initialize_handle(&dh, DFDL_FILE, DFDL_ROOT, 0);
  tpf_dfdl_setData(dh, &buf, sizeof(buf));
  ts_size = strftime(timestamp, sizeof(timestamp), "%Y-%m-%dT%H:%M:%S", ts);
  val.dataType = DFDL_TYPE_STRING;
  val.strCcsid = TPF_CCSID_IBM1047;
  val.length = ts_size;
  val.value.v_pointer = timestamp;
  tpf_dfdl_setVariable(dh, “dateTime", &val, NULL);
  JSONdoc = tpf_dfdl_buildDoc(dh, &docLen, TPF_DFDL_JSON, 0);
}
```

Example – special formatting

	Binary		JSON
Enum:	00000001	⟷	“ECOINN”
TOD clock:	D5B3FD2F 3B3AE8A3	⟷	“02/18/2019 18.46.17”

Example – special formatting

```
<xs:schema xmlns:ufn="http://www.ibm.com/xmlns/prod/ztpf/dfdl/user-functions" >

<xs:group name="hiddenEnum">
   <xs:sequence>
      <xs:element name="BrandEnum" type="xs:unsignedByte" dfdl:lengthKind="explicit"
          dfdl:length="1" dfdl:outputValueCalc="{ufn:BrandNum(../Brand)}" />
   </xs:sequence>
</xs:group>

<xs:sequence dfdl:hiddenGroupRef="hiddenEnum"/>
<xs:element name="Brand" type="xs:string"
          dfdl:inputValueCalc="{ufn:BrandStr(../BrandEnum)}" />
```

Example – special formatting

base/rt/dfdl_ufn.cpp:

```
extern "C" void DFDL_user_function(DFDLHandle dfdlhdl, char *fct_name,
                                   dfdl_data *parm, dfdl_data *result) {
  size_t fctlen = strlen(fct_name);

  switch (fctlen) {
    case 8:
    {
     if (memcmp(fct_name, “BrandNum”, 8) == 0) {
       // Convert string in parm to a number, store in result, and return
     } else if (memcmp(fct_name, “BrandStr”, 8) == 0) {
     }
  }
```

Pointers

Problem

- DFDL handles only contiguous data
- REST interfaces often deal with structures involving multiple variable length strings and variable size arrays.

Pain Points

- Data needs to be copied to a contiguous area of memory to be used by DFDL.

- Variable length strings and variable size areas cause either large memory requirements or structures that are difficult to traverse.

Value Statement

- TPF pointer support for DFDL (PJ45427) provides a way to use non-contiguous data with DFDL.

The Details

The TPF DFDL annotation **indirectKind** and **indirectLength** provides:

- Ability to create and access non-contiguous data with DFDL through the definition of pointers.
- A proof of concept for possible adoption by DFDL 2.0.

Example – pointer definition

Before:

```
typedef struct {
  char module[4];
  char version[3];
  char pathToSO[1023];
  char subPaths[4095];
  char rootDir[1023];
  char timestamp[14];
} tpf_remoteDebugInfoRequest;
```

After:

```
typedef struct {
  char module[4];
  char version[3];
  char pathToSO[1023];
  unsigned short subPathLen;
  char *subPaths;
  char rootDir[1023];
  char timestamp[14];
} tpf_remoteDebugInfoRequest;
```

Example – pointer definition

```
<xs:schema xmlns:tddt="http://www.ibm.com/xmlns/prod/ztpf/xml/lib/tpfattributes" >
<xs:group name="hiddenPathLen">
   <xs:sequence>
      <xs:element name="subPathLen" type="xs:unsignedShort" dfdl:lengthKind="explicit"
           dfdl:length="2" dfdl:lengthUnits="bytes"
           dfdl:outputValueCalc="{dfdl:valueLength(../subPaths,'bytes')}" />
   </xs:sequence>
</xs:group>

<xs:sequence dfdl:hiddenGroupRef="hiddenPathLen"/>
<xs:element name="subPaths" type="xs:string" dfdl:occursCountKind="expression"
           minOccurs="0" maxOccurs="1" dfdl:occursCount="{if (../subPathLen) then 1 else 0}"
           dfdl:lengthKind="explicit" dfdl:length="{../subPathLen}" dfdl:lengthUnits="bytes"
           tddt:indirectKind="pointer" tddt:indirectLength="8" />
```

Recap

PJ45427 provides the ability to:

- Set the length, occurrence, and existence of elements through calculated fields
- Hide elements and create multiple views of data through hidden groups
- Access information not part of the data through DFDL variables
- Create custom transformations through user-exit functions
- Define non-contiguous data through pointers

Some problems introduced were fixed by PJ45615.

Disclaimer

Any reference to future plans are for planning purposes only. IBM reserves the right to change those plans at its discretion. Any reliance on such a disclosure is solely at your own risk. IBM makes no commitment to provide additional information in the future.

Default field exclusion

Problem

- JSON and XML documents built by DFDL contain every field defined which can include many elements and array items where all fields contain a 0 value and do not need to be transmitted.

Pain Points

- Large documents can create larger network bandwidth requirements.
- Updating the DFDL to exclude each optional field and array items is time consuming, error prone, and difficult to maintain.

Value Statement

- DFDL can create smaller JSON and XML documents by excluding elements that contain default values.

The Details

- A new option will be provided for tpf_dfdl_buildDoc in which any elements that contain a default value are excluded.

  For example:

  ```
  JSONDoc = tpf_dfdl_buildDoc(dfdlhdl, &docLen, TPF_DFDL_JSON,
                                                TPF_DFDL_XDFLT);
  ```

- Any empty complex element or trailing empty array items will also be excluded.

Example

Before:

```
{
 “apar”:”PJ45427”,
 “reviewers”:[
    {“name”:”John Smith”,
     “date”:”2018-10-11”},
    {“name”:””,
     “date”:””}
 ],
 “coreqs”:false,
 “miginfo”:true
}
```

After:

```
{
 “apar”:”PJ45427”,
 “reviewers”:[
    {“name”:”John Smith”,
     “date”:”2018-10-11”}
 ],
 “miginfo”:true
}
```

Flatten/unflatten

Problem

- RESTful interfaces often involve many variable length strings and variable size arrays.
- Future direction for handling this type of data is through pointers which simplifies how to traverse the data.
- The data containing pointers may need to be flattened however in order to be filed or moved to shared memory.

Pain Points

- Writing code to handle navigation of variable length fields and variable size arrays takes time (money).
- Flattening and unflattening of non-contiguous data for filing can be expensive to create and maintain.

Value Statement

- DFDL can transform structures with pointers to a flattened representation with offsets cheaply and in a standardized way.

The Details

- A new DFDL API will be provided to either flatten or unflatten structures containing pointer definitions.
- To flatten: a new ECB heap area will be created, 8 or 4 byte addresses will become a corresponding 8 or 4 byte offset from the start of the data area.
- To unflatten: 8 or 4 byte offsets will be changed to corresponding 8 or 4 byte addresses.

Summary

- What's New: PJ45427
  - Calculated fields
  - Hidden groups
  - User-exit functions
  - Pointers
- What's Next:
  - Default field exclusion
  - Flatten/unflatten

Content Survey

ibm.biz/tpf-dfdl

Thank You!

Questions or Comments?

Trademarks

IBM, the IBM logo, ibm.com and Rational are trademarks or registered trademarks of International Business Machines Corp., registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on the Web at "Copyright and trademark information" at www.ibm.com/legal/copytrade.shtml.

Notes

Performance is in Internal Throughput Rate (ITR) ratio based on measurements and projections using standard IBM benchmarks in a controlled environment. The actual throughput that any user will experience will vary depending upon considerations such as the amount of multiprogramming in the user's job stream, the I/O configuration, the storage configuration, and the workload processed. Therefore, no assurance can be given that an individual user will achieve throughput improvements equivalent to the performance ratios stated here.

All customer examples cited or described in this presentation are presented as illustrations of the manner in which some customers have used IBM products and the results they may have achieved. Actual environmental costs and performance characteristics will vary depending on individual customer configurations and conditions.

This publication was produced in the United States. IBM may not offer the products, services or features discussed in this document in other countries, and the information may be subject to change without notice. Consult your local IBM business contact for information on the product or services available in your area.

All statements regarding IBM's future direction and intent are subject to change or withdrawal without notice, and represent goals and objectives only.

Information about non-IBM products is obtained from the manufacturers of those products or their published announcements. IBM has not tested those products and cannot confirm the performance, compatibility, or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

Prices subject to change without notice. Contact your IBM representative or Business Partner for the most current pricing in your geography.

This presentation and the claims outlined in it were reviewed for compliance with US law. Adaptations of these claims for use in other geographies must be reviewed by the local country counsel for compliance with local laws.