DFDL Enhancements
SOA Subcommittee

Bradd Kadlecik
z/TPF Development

Agenda:

Effecient JSON transformations (PJ44767 & PJ45191)

Effecient XML transformations (PJ44894)

Out of order serialization (PJ45191)

Support for BCD (PJ44698)

CSV format and DFDL variables (PJ44894)

Potential DFDL updates

Agenda:

Effecient JSON transformations (PJ44767 & PJ45191)

Effecient XML transformations (PJ44894)

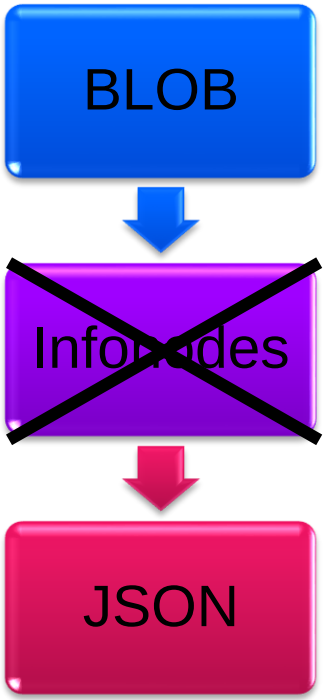
Out of order serialization (PJ45191)

Support for BCD (PJ44698)

CSV format and DFDL variables (PJ44894)

Potential DFDL updates

Efficient JSON creation (PUT14 - PJ44767)



Less of everything!

Less MIPS used
Less ECB heap used
Less function calls to code
And we made the JSON document slightly smaller

REST/Java interfaces updated to use new support.

New DFDL API to build JSON documents directly from binary.

```
char *tpf_dfdl_buildDoc(DFDLHandle dfdlhdl,
                        int *doc_length,
                        DFDLFormat docType,
                        int options);
```

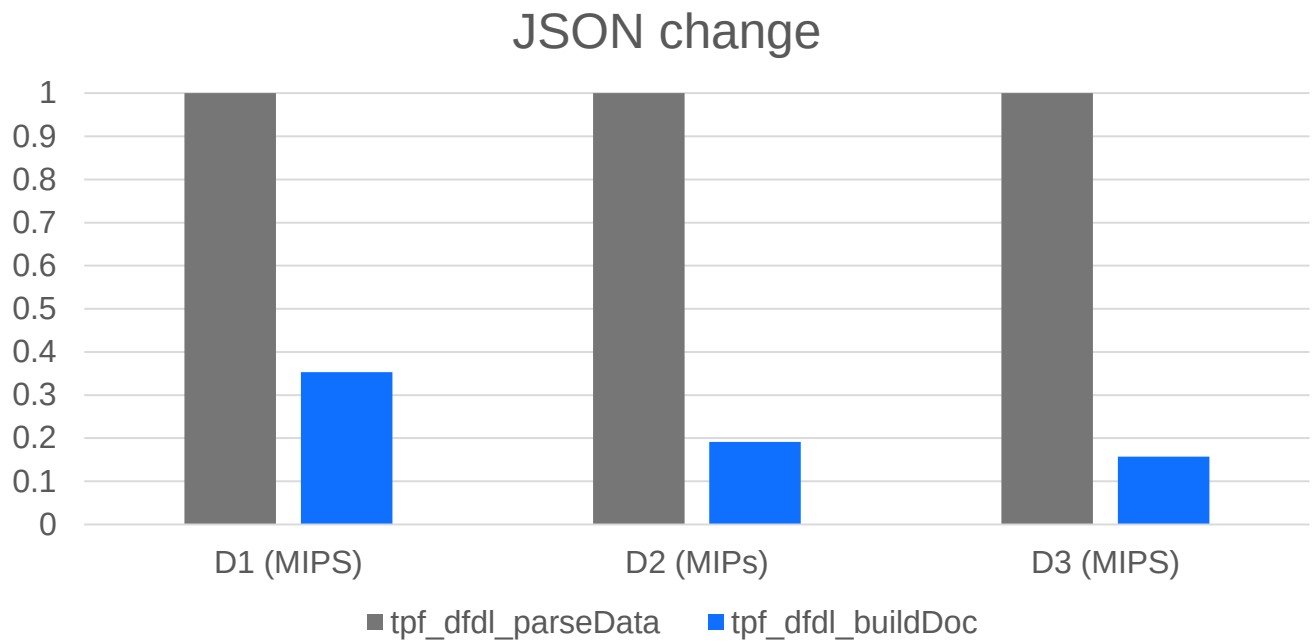

Efficient JSON creation (results may vary)

The MIPS reduction is based mostly on a per element basis.

D1:
227 bytes of data
~50 elements

D2:
4316 bytes of data
~900 elements

D3:
19725 bytes of data
~2000 elements



Efficient JSON creation (coding example)

BLOB -> Infonodes -> JSON

```
tpf_doc_initialize_handle(&xh, NO_PARSER,
                          NULL);
tpf_doc_createJSONstructure(xh,
                          TPF_CCSID_IBM1047,
                          TPF_CCSID_UTF8);

try {
   tpf_dfdl_initialize_handle(&dh, schema_file,
                              root_element, 0);
   tpf_dfdl_setData(dh, buffer, buflen);
   tpf_dfdl_parseData(dh, xh, NULL, 0);
} catch (std::exception &e) {
}

docPtr = tpf_doc_buildDocument(xh, &docLen, 0);
```

BLOB -> JSON

```
try {
   tpf_dfdl_initialize_handle(&dh, schema_file,
                              root_element, 0);
   tpf_dfdl_setData(dh, buffer, buflen);
   docPtr = tpf_dfdl_buildDoc(dh, &docLen,
                              TPF_DFDL_JSON,
                              0);
} catch (std::exception &e) {
}
```


Efficient JSON creation (JSON changes)

tpf_dfdl_parseData, tpf_doc_buildDocument

```
{
   "DFLREC":{
      "lrecLength":“12",
      "lrecKey":"C0",
      "ServiceRecord":{
         "serviceCode":"5",
         “vip”:”true”,
         "numServices":"6",
         "serviceElem":["1","2","3","4","5","6"]
      }
   }
}
```

tpf_dfdl_buildDoc

```
{
   "DFLREC":{
      "lrecLength":12,
      "lrecKey":"C0",
      "ServiceRecord":{
         "serviceCode":5,
         “vip”:true,
         "numServices":6,
         "serviceElem":[1,2,3,4,5,6]
      }
   }
}
```

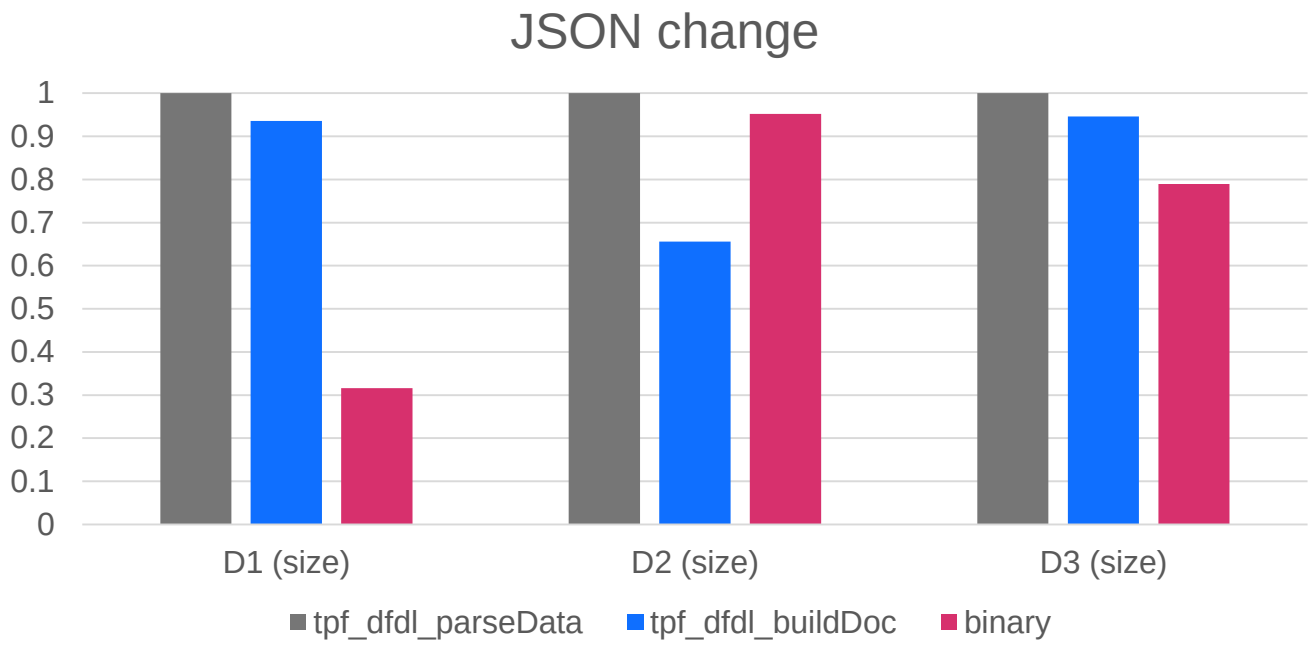

Efficient JSON creation (results may vary)

The data types and organization are the biggest factor in size reduction.

D1:
227 bytes of data
~50 elements

D2:
4316 bytes of data
~900 elements

D3:
19725 bytes of data
~2000 elements



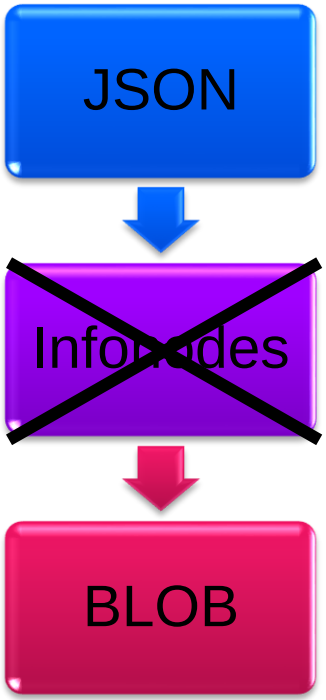
JSON control characters (PUT14 - PJ44894)

```
{
   "stroutput":{
      "e500":"try[&]<>",
      "e1047":"try[&]<>",
      "utf8":"try[&]<>",
      "h500":"testDQ\"\t",
      "h1047":"testNL\n\n",
      "hutf8":"test\r\u0007\f\\“
   }
}
```

PJ44894 added support for handling JSON special characters in strings:
double quote - \”
backslash - \\
backspace - \b
tab - \t
new line - \n
form feed - \f
carriage return - \r

JSON control characters:
\uxxxx

Efficient JSON serialization (PUT15 - PJ45191)



Less MIPS used
Less ECB heap used
Less function calls to code

New DFDL API to serialize JSON documents directly to binary.

REST/Java interfaces updated to use new support.

```
void *tpf_dfdl_serializeDoc(DFDLHandle dfdlhdl,
                            char *document,
                            int doc_length,
                            DFDLFormat docType,
                            unsigned int *data_length,
                            char *start_element,
                            int options);
```

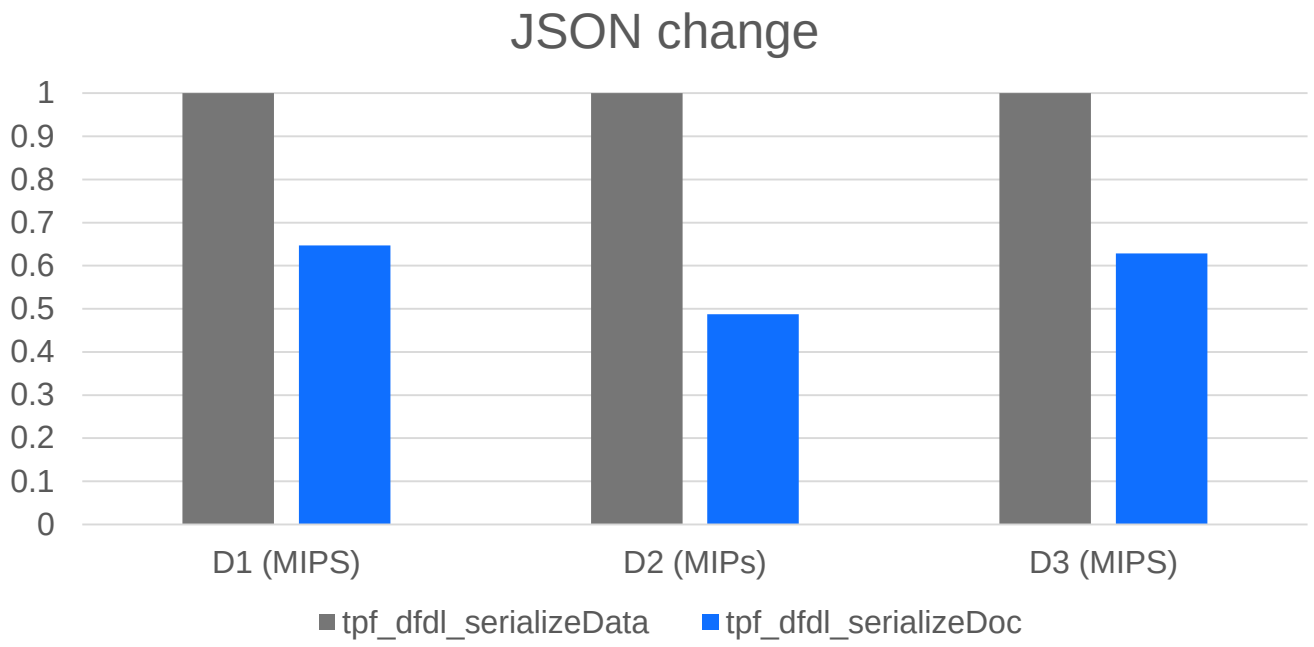

Efficient JSON serialization (results may vary)

There are multiple factors that influence amount of MIPS reduction.

D1:
227 bytes of data
~50 elements

D2:
4316 bytes of data
~900 elements

D3:
19725 bytes of data
~2000 elements



Efficient JSON serialization (coding example)

JSON -> Infonodes -> BLOB

```
tpf_doc_initialize_handle(&xh,
                          B2B_JSON_PARSER,
                          NULL);
tpf_doc_parseDocument(xh, docPtr,
                          TPF_CCSID_IBM1047,
                          docLen, &parse_rc, 0);

try {
   tpf_dfdl_initialize_handle(&dh, schema_file,
                              root_element, 0);
   buffer = tpf_dfdl_serializeData(dh, xh, NULL, 0);
} catch (std::exception &e) {
}

tpf_doc_terminate_handle(&xh);
```

JSON -> BLOB

```
try {
   tpf_dfdl_initialize_handle(&dh, schema_file,
                              root_element, 0);
   buffer = tpf_dfdl_serializeDoc(dh, docPtr, docLen,
                              TPF_DFDL_JSON,
                              &buflen, NULL,
                              0);
} catch (std::exception &e) {
}
```


Agenda:

Effecient JSON transformations (PJ44767 & PJ45191)

Effecient XML transformations (PJ44894)

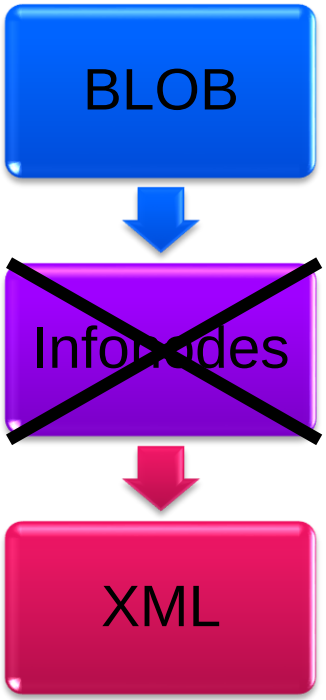
Out of order serialization (PJ45191)

Support for BCD (PJ44698)

CSV format and DFDL variables (PJ44894)

Potential DFDL updates

Efficient XML creation (PUT14 - PJ44894)



Less of everything!

Less MIPS used
Less ECB heap used
Less function calls to code
Smaller XML document
The document is even created in UTF-8!

REST interfaces updated to use new support.

New DFDL API to build XML documents directly from binary.

```
char *tpf_dfdl_buildDoc(DFDLHandle dfdlhdl,
                        int *doc_length,
                        DFDLFormat docType,
                        int options);
```

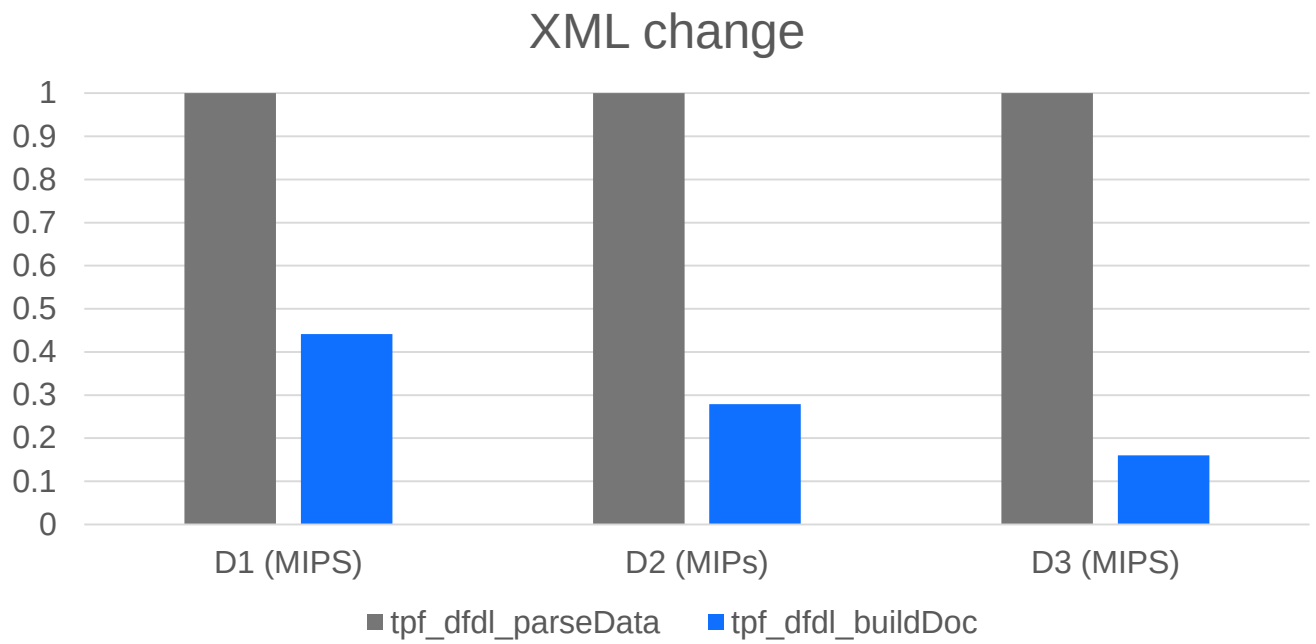

Efficient XML creation (results may vary)

The MIPS reduction is based mostly on a per element basis.

D1:
227 bytes of data
~50 elements

D2:
4316 bytes of data
~900 elements

D3:
19725 bytes of data
~2000 elements



Efficient XML creation (coding example)

BLOB -> Infonodes -> XML

```
tpf_doc_initialize_handle(&xh, NO_PARSER,
                          NULL);
tpf_doc_createXMLstructure(xh, “1.0”,
                   TPF_CCSID_UTF8,
                   TPF_CSNAME_UTF8.
                   STANDALONE_NOT_USED,0);
try {
   tpf_dfdl_initialize_handle(&dh, schema_file,
                              root_element, 0);
   tpf_dfdl_setData(dh, buffer, buflen);
   tpf_dfdl_parseData(dh, xh, NULL, 0);
} catch (std::exception &e) {
}
docPtr = tpf_doc_buildDocument(xh, &docLen, 0);
```

BLOB -> XML

```
try {
   tpf_dfdl_initialize_handle(&dh, schema_file,
                              root_element, 0);
   tpf_dfdl_setData(dh, buffer, buflen);
   docPtr = tpf_dfdl_buildDoc(dh, &docLen,
                              TPF_DFDL_XML,
                              0);
} catch (std::exception &e) {
}
```


Efficient XML creation (tpf_dfdl_parseData, tpf_doc_buildDocument)

```
<?xml version="1.0" encoding="utf-8“ ?>
<ns0:DFLREC xmlns:ns0="http://www.ibm.com/xmlns/prod/ztpf/dfdl/tpfdf/DR26BI">
  <ns0:lrecLength>220</ns0:lrecLength>
  <ns0:lrecKey>C0</ns0:lrecKey>
  <ns0:ServiceRecord>
    <ns0:serviceCode>7</ns0:serviceCode>
    <ns0:extComplexType>
      <drcom:customer xmlns:drcom="http://www.ibm.com/xmlns/prod/ztpf/dfdl/tpfdf/DRCOM">Christopher
Robin</drcom:customer>
      <drcom:age xmlns:drcom=“http://www.ibm.com/xmlns/prod/ztpf/dfdl/tpfdf/DRCOM”>100</drcom:age>
      <drcom:shipto xmlns:drcom="http://www.ibm.com/xmlns/prod/ztpf/dfdl/tpfdf/DRCOM"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:nil="true"/>
    </ns0:extComplexType>
  </ns0:ServiceRecord>
</ns0:DFLREC>
```


Efficient XML creation – 30% smaller document (tpf_dfdl_buildDoc)

```
<?xml version="1.0" encoding="utf-8“?>
<DFLREC xmlns="http://www.ibm.com/xmlns/prod/ztpf/dfdl/tpfdf/DR26BI"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <lrecLength>220</lrecLength>
  <lrecKey>C0</lrecKey>
  <ServiceRecord>
    <serviceCode>7</serviceCode>
    <extComplexType xmlns:ns0="http://www.ibm.com/xmlns/prod/ztpf/dfdl/tpfdf/DRCOM">
      <ns0:customer>Christopher Robin</ns0:customer>
      <ns0:age>100</ns0:age>
      <ns0:shipto xsi:nil="true"/>
    </extComplexType>
  </ServiceRecord>
</DFLREC>
```

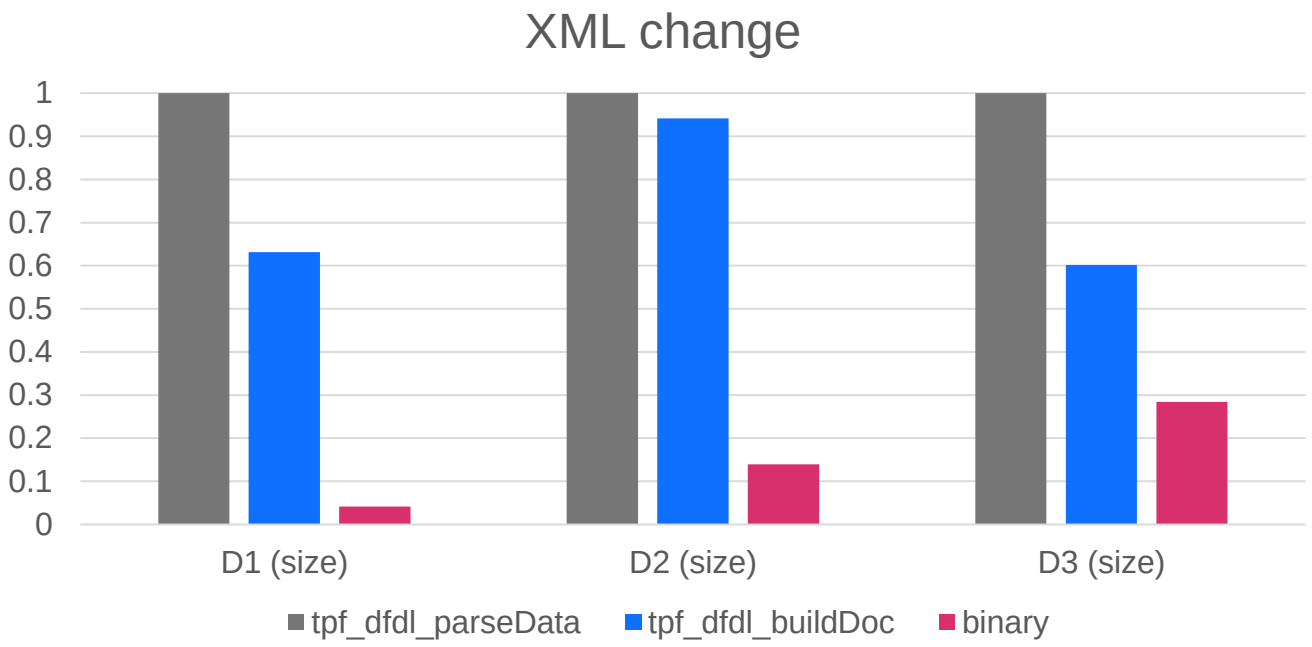

Efficient XML creation (results may vary)

The DFDL definition is the biggest factor in size reduction.

D1:
227 bytes of data
~50 elements

D2:
4316 bytes of data
~900 elements

D3:
19725 bytes of data
~2000 elements



XML control characters (PUT14 - PJ44894)

```
<?xml version="1.0" encoding="utf-8"?>
<stroutput
xmlns="http://www.ibm.com/xmlns/prod/ztpf/dfdl/gen
/stroutput"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance">
  <e500>try[&amp;]&lt;&gt;</e500>
  <e1047>try[&amp;]&lt;&gt;</e1047>
  <utf8>try[&amp;]&lt;&gt;</utf8>
  <h500>testDQ"&#9;</h500>
  <h1047>testNL&#10;&#133;</h1047>
  <hutf8>test&#13;&#7;&#12;\</hutf8>
</stroutput>
```

PJ44894 added support for handling XML special characters in strings:
ampersand - &
greater than - >
less than - <

XML control characters:
&#nnn;

Agenda:

Effecient JSON transformations (PJ44767 & PJ45191)

Effecient XML transformations (PJ44894)

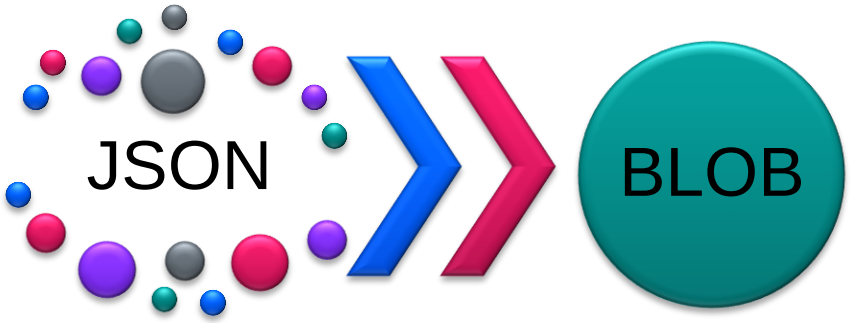
Out of order serialization (PJ45191)

Support for BCD (PJ44698)

CSV format and DFDL variables (PJ44894)

Potential DFDL updates

Out of order serialization (PJ45191)



Unordered Ordered

JSON does not dictate an order of elements.

tpf_dfdl_serializeData was updated to support any order of elements.

tpf_dfdl_serializeDoc requires the order of elements to match the order defined in DFDL.

Serializing unordered elements will be less efficient than ordered.

Out of order serialization (binary order)

```
struct stdhd                        /* TPF's standard header          */
{
  unsigned char stdbid[2];          /* Record ID                      */
  unsigned char stdchk;             /* Record code                    */
  unsigned char stdctl;             /* Data control                   */
  unsigned char stdpgm[4];          /* Program ID                     */
  unsigned int  stdfch;             /* Forward Chain Field            */
  unsigned int  stdbch;             /* Backward Chain Field           */
}
```


Out of order serialization (JSON example)

Ordered JSON:

```
{
   “stdhd”: {
      “stdbid”: "BD",
      “stdchk”: 1,
      “stdctl”: 0,
      “stdpgm”: "ABCD",
      “stdfch”: 0,
      “stdbch”: 0
   }
}
```

Unordered JSON:

```
{
   “stdhd”: {
      “stdchk”: 1,
      “stdbid”: "BD",
      “stdpgm”: "ABCD",
      “stdctl”: 0,
      “stdbch”: 0,
      “stdfch”: 0
   }
}
```

Binary:

```
C2C40100C1C2C3C4
0000000000000000
```


Out of order serialization (coding example)

Unordered JSON -> BLOB

```
tpf_doc_initialize_handle(&xh,
                          B2B_JSON_PARSER,
                          NULL);
tpf_doc_parseDocument(xh, docPtr,
                          TPF_CCSID_IBM1047,
                          docLen, &parse_rc, 0);

try {
   tpf_dfdl_initialize_handle(&dh, schema_file,
                              root_element, 0);
   buffer = tpf_dfdl_serializeData(dh, xh, NULL, 0);
} catch (std::exception &e) {
}

tpf_doc_terminate_handle(&xh);
```

Ordered JSON -> BLOB

```
try {
   tpf_dfdl_initialize_handle(&dh, schema_file,
                              root_element, 0);
   buffer = tpf_dfdl_serializeDoc(dh, docPtr, docLen,
                              TPF_DFDL_JSON,
                              &buflen, NULL,
                              0);
} catch (std::exception &e) {
}
```


Agenda:

Effecient JSON transformations (PJ44767 & PJ45191)

Effecient XML transformations (PJ44894)

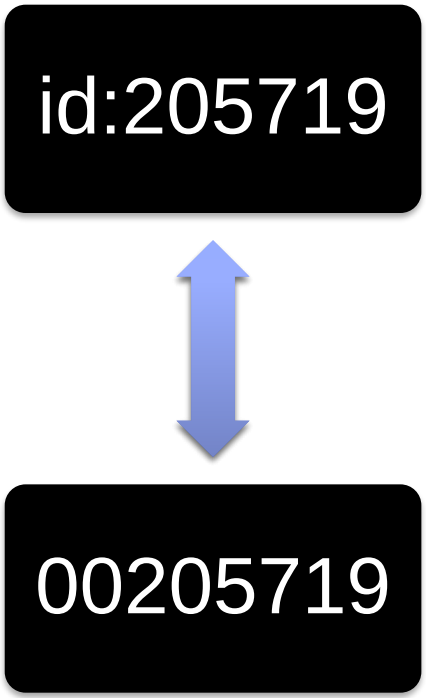
Out of order serialization (PJ45191)

Support for BCD (PJ44698)

CSV format and DFDL variables (PJ44894)

Potential DFDL updates

Support for Binary Coded Decimal (PJ44698)



Binary Coded Decimals can be defined by setting the DFDL binaryNumberRep attribute to “bcd” for decimal or non-negative integer types.

DFDL example:

<xs:element name=“id” type=“xs:unsignedInt” dfdl:binaryNumberRep=“bcd” dfdl:lengthKind=“explicit” dfdl:length=“4”/>

Agenda:

Effecient JSON transformations (PJ44767 & PJ45191)

Effecient XML transformations (PJ44894)

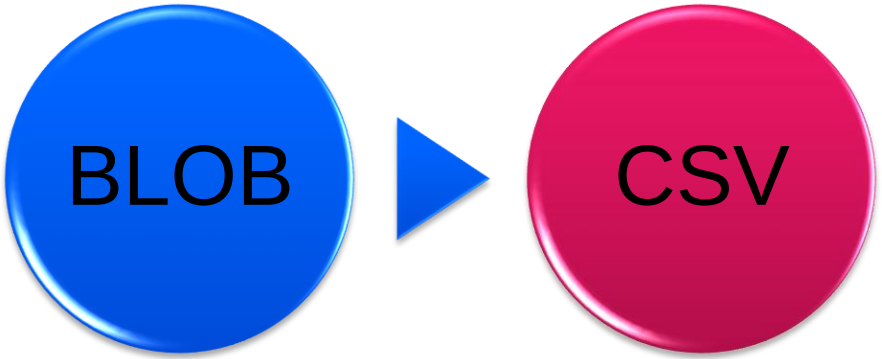
Out of order serialization (PJ45191)

Support for BCD (PJ44698)

CSV format and DFDL variables (PJ44894)

Potential DFDL updates

CSV transformation (PUT14 - PJ44894)



New DFDL API can also build CSV documents directly from binary.

```
char *tpf_dfdl_buildDoc(DFDLHandle dfdlhdl,
                        int *doc_length,
                        DFDLFormat docType,
                        int options);
```

docType = TPF_DFDL_CSV

Since this is not officially supported, it is not found in the knowledge center.

It might not create what you want for complex data.

CSV transformation (API options)

Binary: stdhd

```
C2C40100C1C2C3C4
0000000000000000
```

Default options:

stdbid,stdchk,stdctl,stdpgm,stdfch,stdbch
BD,1,0,ABCD,0,0

TPF_DFDL_XTAGS:

BD,1,0,ABCD,0,0

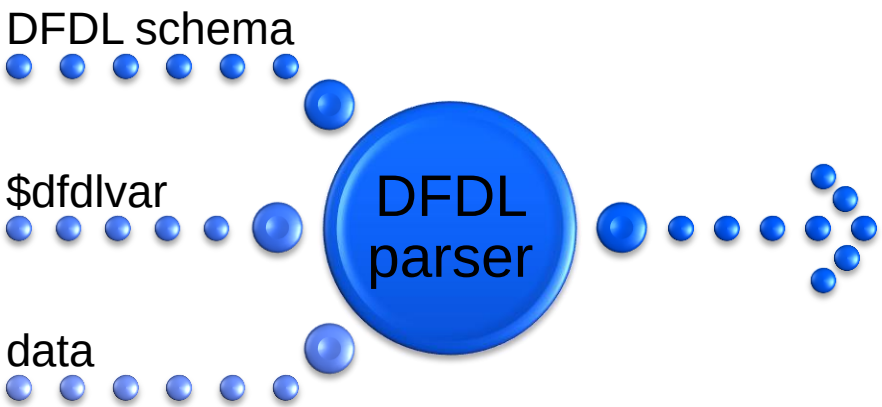
TPF_DFDL_DQSTR:

stdbid,stdchk,stdctl,stdpgm,stdfch,stdbch
“BD”,1,0,”ABCD”,0,0

TPF_DFDL_ROW_VAL:

stdbid,BD
stdchk,1
stdctl,0
stdpgm,ABCD
stdfch,0
stdbch,0

DFDL Expression Variables (PUT14 - PJ44894)



DFDL variables allow a way to programmatically modify the output document using information not contained within the binary data.

New unofficial DFDL API can set the value for a DFDL variable referenced by any DFDL expression.

```
void tpf_dfdl_setVariable(DFDLHandle dfdlhdl,
                          char *varName,
                          dfdl_data *value);
```

Official support will likely modify this API to add a namespace parameter.

DFDL Expression Variables (DFDL example)

Can use DFDL variables to selectively exclude certain fields for security reasons.

Element “ssn” is included if DFDL variable “spi” is 1.

Else an empty sequence of the same length (by using trailingSkip) is used.

```
<xs:choice>
   <xs:element name=“ssn” type=“xs:unsignedInt”
                  dfdl:lengthKind=“explicit” dfdl:length=“4”>
      <xs:annotation>
         <xs:appinfo source="http://www.ogf.org/dfdl/">
            <dfdl:discriminator>{$spi eq 1}</dfdl:discriminator>
         </xs:appinfo>
      </xs:annotation>
   </xs:element>
   <xs:sequence dfdl:trailingSkip=“4”/>
</xs:choice>
```


DFDL Expression Variables (coding example)

Element “ssn” excluded from JSON

```
try {
   tpf_dfdl_initialize_handle(&dh, schema_file,
                              root_element, 0);
   tpf_dfdl_setData(dh, buffer, buflen);
   docPtr = tpf_dfdl_buildDoc(dh, &docLen,
                              TPF_DFDL_JSON,
                              0);
} catch (std::exception &e) {
}
```

Element “ssn” included in JSON

```
try {
   tpf_dfdl_initialize_handle(&dh, schema_file,
                              root_element, 0);
   tpf_dfdl_setData(dh, buffer, buflen);
   val.dataType = DFDL_TYPE_UINT;
   val.value.v_ulong = 1;
   tpf_dfdl_setVariable(dh, “spi”, &val);
   docPtr = tpf_dfdl_buildDoc(dh, &docLen,
                              TPF_DFDL_JSON,
                              0);
} catch (std::exception &e) {
}
```


Let us know if you have use cases for either CSV format or DFDL variables so it can be officially supported.

Agenda:

Effecient JSON transformations (PJ44767 & PJ45191)

Effecient XML transformations (PJ44894)

Out of order serialization (PJ45191)

Support for BCD (PJ44698)

CSV format and DFDL variables (PJ44894)

Potential DFDL updates

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.

Calculated field values

DFDL provides the outputValueCalc property to set a field value during serialization based on a DFDL expression. This provides:

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

Thank You!

Questions or Comments?

Trademarks

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.