DFDL Enhancements

SOA Subcommittee

Bradd Kadlecik
z/TPF Development

IBM **z/TPF**
April 4th, 2017

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.

Agenda

- Brief Intro
- What's New
- What's Next

What is DFDL?

- Data Format Description Language (not related to TPFDF)
- A standardized way of describing data.
- Allows for data transformation services both on and off z/TPF. For example, data can be transformed from native format to XML, JSON, BSON, etc and vice versa.

DFDL Enablement:

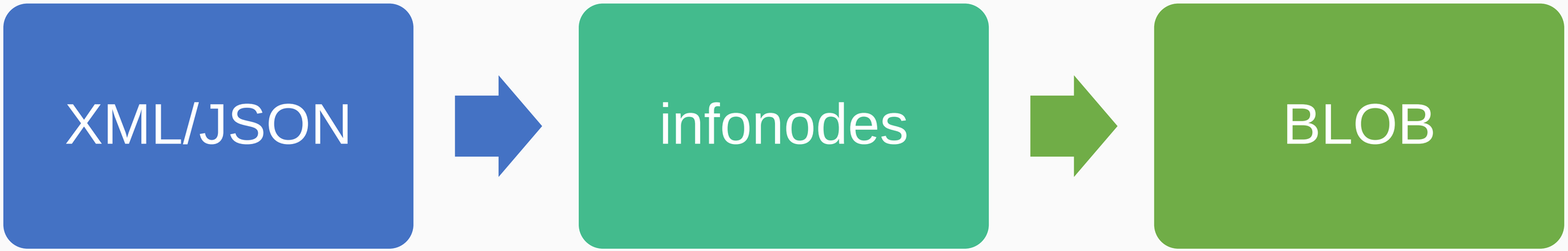
- Data Events (2014)
  - transform data to XML or JSON
- z/TPF support for MongoDB (2015)
  - transform data to/from BSON
- DFDL Serializer (2016)
  - transform XML or JSON to data
- REST/Java interfaces (2017)

DFDL Serializer (PJ44104)

The z/TPF parser APIs in conjunction with the DFDL serialize API can be used to easily convert XML or JSON documents to a data stream.

XML/JSON serialization

tpf _doc_parseDocument



tpf_dfdl_serializeData

JSON -> binary

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

```
C2C40100C1C2C3C4
00000000 00000000
```

XML Serialization (EBCDIC)

```
XMLHandle xh;
DFDLHandle dh;
rc = tpf_doc_initialize_handle(&xh, B2B_XML_SCANNER, NULL);
rc = tpf_doc_parseDocument(xh, XMLdoc, TPF_CCSID_UTFEBCDIC, docLen,
                                                    &parse_rc, 0);
try {
    tpf_dfdl_initialize_handle(&dh, “stdhd.gen.dfdl.xsd”, “stdhd”, 0);
    struct stdhd *hdrInfo = (struct stdhd *) tpf_dfdl_serializeData(dh, xh, NULL, 0);
} catch (std::exception &e) {
}
```

JSON Serialization (UTF-8)

```
XMLHandle xh;
DFDLHandle dh;
rc = tpf_doc_initialize_handle(&xh, B2B_JSON_PARSER, NULL);
rc = tpf_doc_parseDocument(xh, JSONdoc, TPF_CCSID_IBM1047, docLen,
                                                    &parse_rc, 0);
try {
    tpf_dfdl_initialize_handle(&dh, "stdhd.gen.dfdl.xsd", "stdhd", 0);
    struct stdhd *hdrInfo = (struct stdhd *) tpf_dfdl_serializeData(dh, xh, NULL, 0);
} catch (std::exception &e) {
}
```

Data buffer allocation

DFDL serializer uses ECB heap if no data buffer is associated with the DFDL handle.

```
tpf_dfdl_initialize_handle(&dh, "stdhd.gen.dfdl.xsd", "stdhd", 0);
struct stdhd *hdrInfo = (struct stdhd *) tpf_dfdl_serializeData(dh, xh, NULL, 0);
```

Alternatively, a data buffer address can be provided.

```
struct stdhd hdrInfo;
memset(&hdrInfo, 0, sizeof(hdrInfo));
tpf_dfdl_initialize_handle(&dh, "stdhd.gen.dfdl.xsd", "stdhd", 0);
tpf_dfdl_setData(dh, &hdrInfo, sizeof(hdrInfo));
tpf_dfdl_serializeData(dh, xh, NULL, 0);
```

Start serializing anywhere

The XML/JSON handle's position pointer is used to determine where to begin at.

{"inmsg":
  {"msgInfo": { .... },
   "msgdata": { .... }
  }
}

tpf_doc_positionBeforeElementTagName(**xh**, "msgdata");
tpf_dfdl_serializeData(dh, **xh**, NULL, 0);

Subset serialization

An absolute XPath is used to set which piece to serialize within the DFDL description.

```
struct big_struct {
    struct struct1 s1;
    struct struct2 s2;
    ...
} workarea;

tpf_dfdl_initialize_handle(&dh, “big_struct.gen.dfdl.xsd”, “big_struct”, 0);
tpf_dfdl_setData(dh, &workarea, sizeof(struct big_struct));
tpf_dfdl_serializeData(dh, xh, “/big_struct/s2”, 0);
```

XML/JSON Considerations

XML attributes are not supported by DFDL and are ignored during serialization.

<label attr="val">label_val</label>

JSON elements must maintain order (out of order sequences not yet supported) &

JSON does not require the "root element" name for serialization.

```
{"stdhd":
    {"stdbid":"BD", .... }
}
```

- is the same as -

{"stdbid":"BD", ... }

Future Tentative Plans

- Generate DFDL for assembler DSECTs via maketpf
- Support binary coded decimal (BCD) format
- Greatly improve performance of JSON transformation

DFDL generator for DSECTs

- Use same maketpf "dfdl" target to generate DFDL for either assembler or C/C++
  - maketpf PRG1 dfdl
- Use maketpf TPF_DFDL_DIR environment variable to control where files are written
- Generate DFDL for all referenced DSECTs

Support BCD format

Binary Coded Decimal numbers can be defined by setting the DFDL **binaryNumberRep** attribute to "**bcd**" for decimal and integer types.

"1234"

00001234

Improve JSON conversion

New APIs to convert directly from a data stream to JSON and vice versa

- REST and Java linkage improved with no application changes

THANK YOU

Questions or comments?

Bradd Kadlecik
z/TPF Development

IBM **z/TPF**
April 4th, 2017

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.