z/TPF Detailed Summary

z/TPF Middleware - DFDL

Bradd Kadlecik
z/TPF Development

Agenda

- Intro to DFDL
- Creating DFDL schemas
- Customizing DFDL schemas
- Using DFDL

Agenda

- Intro to DFDL
- Creating DFDL schemas
- Customizing DFDL schemas
- Using DFDL

Why Data Format Description Language (DFDL)?

- There are a number of data formats used today including some more popular ones like XML and JSON.
- There was no universal standard for modeling data (both text and binary).
- A number of products created their own proprietary method for modeling data which prevented integration across the products.
- TPF examples: Debugger XML files, business events TPF Data Model.
- “Unlocking” data requires being able to model data in a standard way so that it can be converted to or from various data formats (JSON, XML, CSV, BSON, Java properties, etc) as needed.

What is Data Format Description Language (DFDL)?

- DFDL is to C structures and assembler DSECTs what XML/JSON is to data.
- A universal, shareable, non-prescriptive description for general text and binary data formats.
- An open standard from the Open Grid Forum.
- NOT a data format.
- DFDL uses a subset of XML schema which describes the **logical format** of the data and adds DFDL annotations to describe the **native format**.
- DFDL is **valid** XML schema that can also describe the XML format in addition to relative equivalencies in other formats.

Example: Native Format description

ASSEMBLER DSECT

```
STDHDR DS 0CL16    TPF’s STANDARD HEADER
STDBID DS CL2      FILE ID
STDCHK DS X        BLOCK CHECK CHARACTER
STDCTL DS B        CONTROL BYTE
STDPGM DS F        LAST FILING PROGRAM
STDFCH DS F        FORWARD CHAIN ADDRESS
STDBCH DS F        BACKWARD CHAIN ADDRESS
```

C STRUCTURE

```
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  */
  unsigned int  stdbch;    /* Backward Chain */
}
```

Example: Native Format

```
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  */
  unsigned int  stdbch;    /* Backward Chain */
}
```

```
0000    C2C40100C1C2C3C4
0008    0000000000000000
```

XML Schema: Logical Format description

```
<xs:element name="stdhd">
   <xs:complexType>
      <xs:sequence>
         <xs:element name="stdbid" type="xs:hexBinary"/>

         <xs:element name="stdchk" type="xs:unsignedByte"/>

         <xs:element name="stdctl" type="xs:hexBinary"/>

         <xs:element name="stdpgm" type="xs:string"/>

         <xs:element name="stdfch" type="xs:unsignedInt"/>

         <xs:element name="stdbch" type="xs:unsignedInt"/>

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

Example: XML format

```
<?xml version="1.0" encoding="utf-8"?>
<stdhd xmlns="http://www.ibm.com/xmlns/prod/ztpf/dfdl/gen/stdhd">
    <stdbid>C2C4</stdbid>
    <stdchk>1</stdchk>
    <stdctl>00</stdctl>
    <stdpgm>ABCD</stdpgm>
    <stdfch>0</stdfch>
    <stdbch>0</stdbch>
</stdhd>
```

DFDL: Native & Logical Format description

```
<xs:element name="stdhd" dfdl:lengthKind="implicit">
   <xs:complexType>
      <xs:sequence>
         <xs:element name="stdbid" type="xs:hexBinary"
                     dfdl:length="2"/>
         <xs:element name="stdchk" type="xs:byte"
                     dfdl:length="1"/>
         <xs:element name="stdctl" type="xs:hexBinary"
                     dfdl:length="1"/>
         <xs:element name="stdpgm" type="xs:string"
                     dfdl:length="4"/>
         <xs:element name="stdfch" type="xs:unsignedInt"
                     dfdl:length="4"/>
         <xs:element name="stdbch" type="xs:unsignedInt"
                     dfdl:length="4"/>
      </xs:sequence>
   </xs:complexType>
</xs:element>
```

Example: XML format vs binary

```
<?xml version="1.0" encoding="utf-8"?>
<stdhd
xmlns="http://www.ibm.com/xmlns/prod/zt
pf/dfdl/gen/stdhd">
    <stdbid>C2C4</stdbid>
    <stdchk>1</stdchk>
    <stdctl>00</stdctl>
    <stdpgm>ABCD</stdpgm>
    <stdfch>0</stdfch>
    <stdbch>0</stdbch>
</stdhd>
```

```
C2C40100C1C2C3C4
0000000000000000
```

Example: JSON vs XML format

```
{
 stdhd: {
  stdbid: "C2C4",
  stdchk: 1,
  stdctl: 00,
  stdpgm: "ABCD",
  stdfch: 0,
  stdbch: 0
 }
}
```

```
{
  stdbid: "C2C4",
  stdchk: 1,
  stdctl: 00,
  stdpgm: "ABCD",
  stdfch: 0,
  stdbch: 0
}
```

```
<?xml version="1.0" encoding="utf-8"?>
<stdhd
xmlns="http://www.ibm.com/xmlns/prod/z
tpf/dfdl/gen/stdhd">
    <stdbid>C2C4</stdbid>
    <stdchk>1</stdchk>
    <stdctl>00</stdctl>
    <stdpgm>ABCD</stdpgm>
    <stdfch>0</stdfch>
    <stdbch>0</stdbch>
</stdhd>
```

How can it be used on z/TPF today?

- Business 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)
  - transform data to/from REST interface via OpenAPI (swagger)
- creating CSV documents

Agenda

- Intro to DFDL
- Creating DFDL schemas
- Customizing DFDL schemas
- Using DFDL

Creating DFDL for z/TPFDF records

ZUDFM DESCRIPTOR command:

- Uses MLS information to create DFDL information.
- Creates a DFDL file (.tpfdf.dfdl.xsd extension) in UTF-8 encoding to ftp in binary mode.

Example:

```
zudfm descr file-dr26bi
CSMP0097I 10.51.41 CPU-B SS-BSS SSU-HPN IS-01
UDFM0561I 10.51.41 DR26BI FILE DESCRIPTOR BUILD STARTED
/etc/ztpfdf/descr/DR26BI.tpfdf.dfdl.xsd
UDFM0562I 10.51.41 DR26BI FILE DESCRIPTOR BUILD COMPLETE.
FILES CREATED.
```

z/TPF DFDL generation utility

- Provides ability to create DFDL for C structures
- The tpfdfdlgen utility is shipped with z/TPF and built in linux/bin
- Maketpf contains a “dfdl” target to generate DFDL
  ex: maketpf qzz9 dfdl
- Output is written to TPF_DFDL_DIR in maketpf.cfg.
- A DFDL file is created for each structure
- Generated file names have the format of:
  <struct or typedef name>.gen.dfdl.xsd
- Additional options are provided by TPF_DFDL_OPTS.
  -p (generate information for pointers)

Sample setup for generating DFDL

maketpf.cfg

```
TPF_DFDL_DIR := ~/sample/dfdl
```

qzz2.c

```
#include <tpf/c_mfst.h>
void QZZ2 (struct mf1sec x)
{
}
```

DFDL generation example

braddk@linuxtpf:~/sample> **maketpf qzz2 dfdl**

tpfdfdlgen /home/braddk/sample/temp/obj/qzz2.o -d ~/sample/dfdl

DFDLGEN0015I: DFDL files generated in /home/braddk/sample/dfdl/

braddk@linuxtpf:~/sample> ls dfdl

dctmio.gen.dfdl.xsd mf1sec.gen.dfdl.xsd

mf0sec.gen.dfdl.xsd tpf_TOD_type.gen.dfdl.xsd

Mapping C to DFDL

C	DFDL
struct mf1sec;	<xs:group name="mf1sec">
typedef struct mf1sec __mf1sec_t;	<xs:complexType name="__mf1sec_t">
typedef unsigned int uint;	<xs:simpleType name="uint">
union	<xs:choice>
unsigned char mf1liv;	<xs:element name="mf1liv" dfdl:length="1" type="xs:unsignedByte" />
unsigned char mf1psns[3];	<xs:element name="mf1psns" dfdl:length="3" type="xs:string" />
unsigned int _filler46[3]; /* Spare */	<xs:element name="_filler46" dfdl:length="4" type="xs:unsignedInt" minOccurs="3" maxOccurs="3" />

Agenda

- Intro to DFDL
- Creating DFDL schemas
- Customizing DFDL schemas
- Using DFDL

DFDL customizations – IBM Knowledge Center

Change the targetNamespace

Create meaningful element names

Verify generated data types

Create discriminators for conditional data

Create expressions for variable length fields

Create expressions for variable size arrays

Determine required versus not required

Hide elements

Create pointer references

z/TPF DFDL Filename Convention

Format: **<filename>.<resource type>.dfdl.xsd**

DFDL filenames end in .xsd (XML schema definition) to make them easily recognizable to XML or DFDL editors.

The file extension of .dfdl.xsd allows common deployment to recognize it as a DFDL file.

Examples of **<resource type>:**

- tpfdf: TPFDF definitions
- lib: IBM commonly referenced definitions
- de/se: data event/signal event message

z/TPF Namespace Convention for DFDL

The targetNamespace is the unique identifier for all schema components in the current document.

z/TPF namespace:
http://www.ibm.com/xmlns/prod/ztpf

DFDL filenames are appended in reverse order to create a unique namespace per filename.

- PNR.tpfdf.dfdl.xsd becomes:
  dfdl/tpfdf/PNR
- Resultant namespace:
  http://www.ibm.com/xmlns/prod/ztpf/dfdl/tpfdf/PNR

DFDL Expressions

DFDL uses a subset of XPath expressions.

DFDL expressions are enclosed within curly braces: “{“ “}”

Only the following DFDL annotations/attributes can contain DFDL expressions:

- dfdl:length - used for variable length elements (parse only)
- dfdl:occursCount - used for variable size arrays (parse only)
- dfdl:discriminator - used for variable layouts (parse only)
- dfdl:inputValueCalc - used for element values not found in the data (parse only)
- dfdl:outputValueCalc - used for element values not found in the document (unparse only)
- dfdl:encoding - used for conditional string encoding
- dfdl:terminator - used for delimited length elements

Agenda

- Intro to DFDL
- Creating DFDL schemas
- Customizing DFDL schemas
- Using DFDL

Loading DFDL schemas to z/TPF

DFDL is loaded to z/TPF using common deployment, making the DFDL information accessible to the system (DFDL APIs, Business Events, MongoDB, etc).

- All DFDL files must have the following file exension: **.dfdl.xsd**
- All DFDL files must be loaded to the following location on z/TPF: **/sys/tpf_pbfiles/tpf-fdes**

DFDL files are automatically deployed by common deployment.

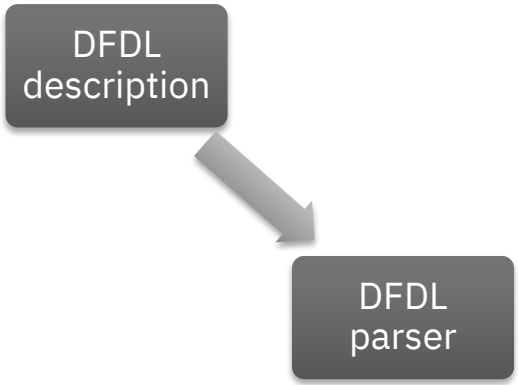
Files are verified and active during loadset activation.

DFDL changes can be loaded with program changes to keep structure changes in synch.

Creating JSON from DFDL

```
#include <tpf/cdfdl.h>      /* DFDL APIs */
#include <exception>        /* DFDL error return */

DFDLHandle dh = 0;          /* handle for DFDL parser */
char *docPtr;               /* JSON document output */
int docLen = 0;             /* JSON document size */
struct mydata buf;          /* data to process to JSON */

#define DFDL_FILE "mydata.gen.dfdl.xsd“
#define DFDL_ROOT "mydata“

...

try{
   tpf_dfdl_initialize_handle(&dh, DFDL_FILE, DFDL_ROOT, 0);
   tpf_dfdl_setData(dh, &buf, sizeof(buf));
   docPtr = tpf_dfdl_buildDoc(dh, &docLen, TPF_DFDL_JSON, 0);
} catch (std::exception &e) {
   // error message in e.what()
}

if (dh) tpf_dfdl_terminate_handle(dh);
```

Identify the DFDL description (tpf_dfdl_initialize_handle)

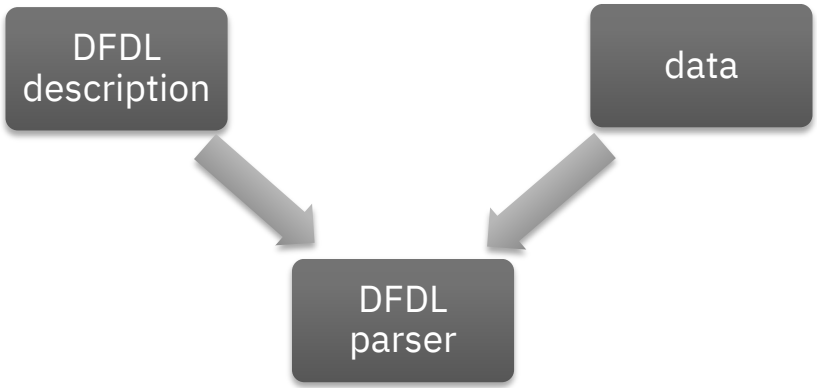


Identify the DFDL description

tpf_dfdl_initialize_handle(DFDLHandle *dfdlhdl, char *schema_file, char *root_element, int options);

1. dfdlhdl - A pointer to the location to store the DFDL API handle that is associated with the DFDL structure to be processed. This function initializes the handle and allocates the required ECB heap storage for the handle.
2. schema_file - A pointer to the name of the DFDL schema file that was loaded to the /sys/tpf_pbfiles/tpf-fdes directory by using common deployment.
3. root_element - A pointer to the name of a complex element within the DFDL schema file that defines the binary data to be processed. The root element denotes the starting and ending point of the binary data.
4. options - Specifies the special processing options for this function.

Identify the data to process (tpf_dfdl_setData)

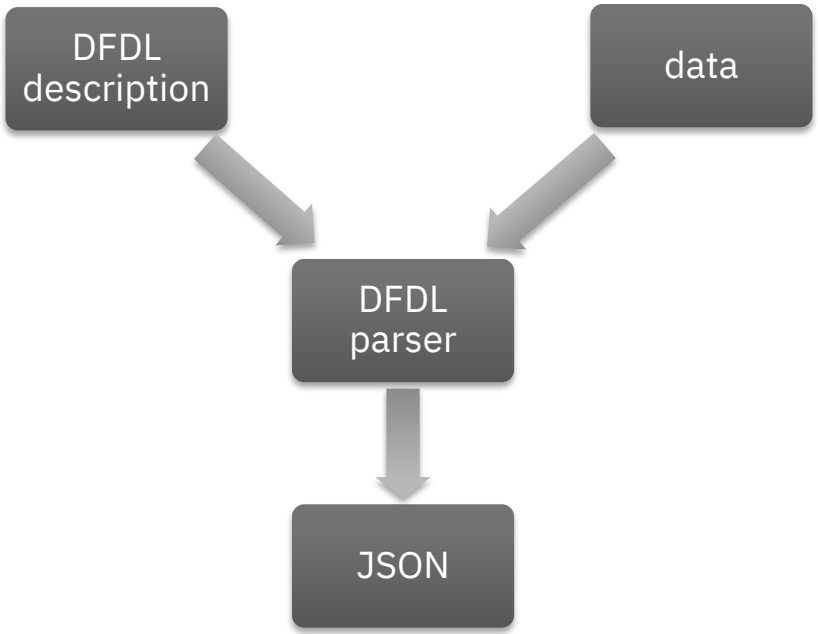


Identify the data to process

tpf_dfdl_setData(DFDLHandle dfdlhdl, void *buffer, unsigned int length);

1. dfdlhdl - A DFDL API handle associated with the DFDL structure that describes the binary data.
2. buffer - A pointer to a data area that is described by a DFDL schema.
3. length - The length of the data stream that is provided by the buffer parameter. This parameter specifies the maximum length of the binary data. This value is used to ensure that computed element offsets and data accesses (for computed offsets) are within the buffer area; otherwise, a std::length_error exception is issued. If the value is 0, this validation is not performed.

Create the document



Create the document

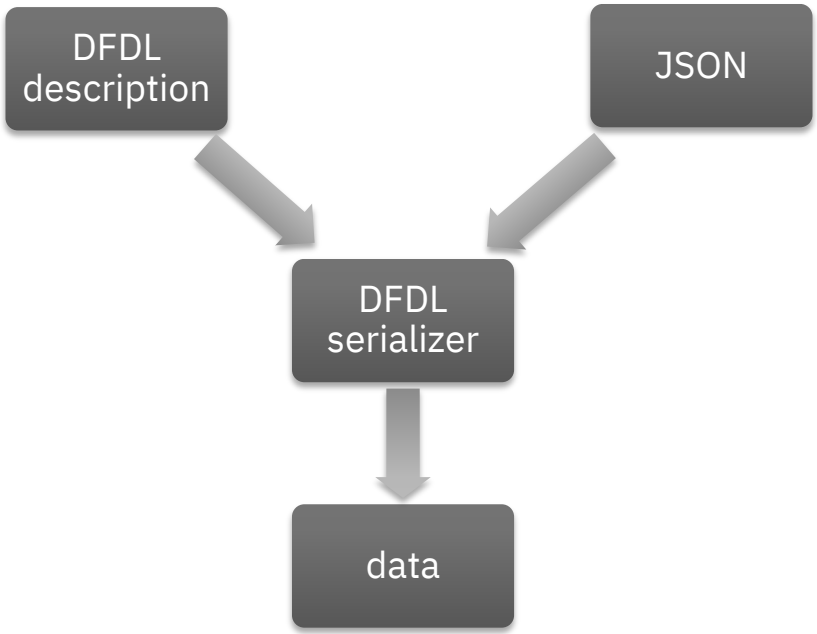
There exists 2 methods of creating a document using DFDL:

1. Create the document directly (**simpler, higher performance**) :
   char ***tpf_dfdl_buildDoc**(DFDLHandle dfdlhdl, int *doc_length, DFDLFormat docType, int options);

2. Create the document using the z/TPF parser APIs (older) :
   void **tpf_dfdl_parseData**(DFDLHandle dfdlhdl, XMLHandle api_handle, char *doc_ElementTagName, int option);

   char ***tpf_doc_buildDocument**(XMLHandle api_handle, int *doc_length, int processing_ind);

Serialize the data



Serialize the data

There exists 2 methods of serializing the data using DFDL:

1. Serialize the data directly (**higher performance but requires order**) :
   char ***tpf_dfdl_serializeDoc**(DFDLHandle dfdlhdl, char *document, int doc_length, DFDLFormat docType, unsigned int *data_length, char *start_element, int options);

2. Serialize the data using the z/TPF parser APIs (**allows any order**) :
   int **tpf_doc_parseDocument**(XMLHandle api_handle, char *document, tpf_ccsid_t CCSID, int docLength, int *parserRetCode, int flags);

   void ***tpf_dfdl_serializeData**(DFDLHandle dfdlhdl, XMLHandle api_handle, char *root_element, int options);

References

DFDL tutorials created by the DFDL Working Group at the Open Grid Forum

- http://redmine.ogf.org/dmsf/dfdl-wg?folder_id=5485

DFDL developerWorks tutorials

- http://ibm.biz/startdfdl

DFDL specification reference

- http://www.ogf.org/dfdl

Thank You!

Questions or Comments?

TPF Challenge

Try the DFDL part of the challenge at

http://ibm.biz/tpfchallenge

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.