DFDL Enhancements
PJ46213

Bradd Kadlecik

IBM Z

IBM

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

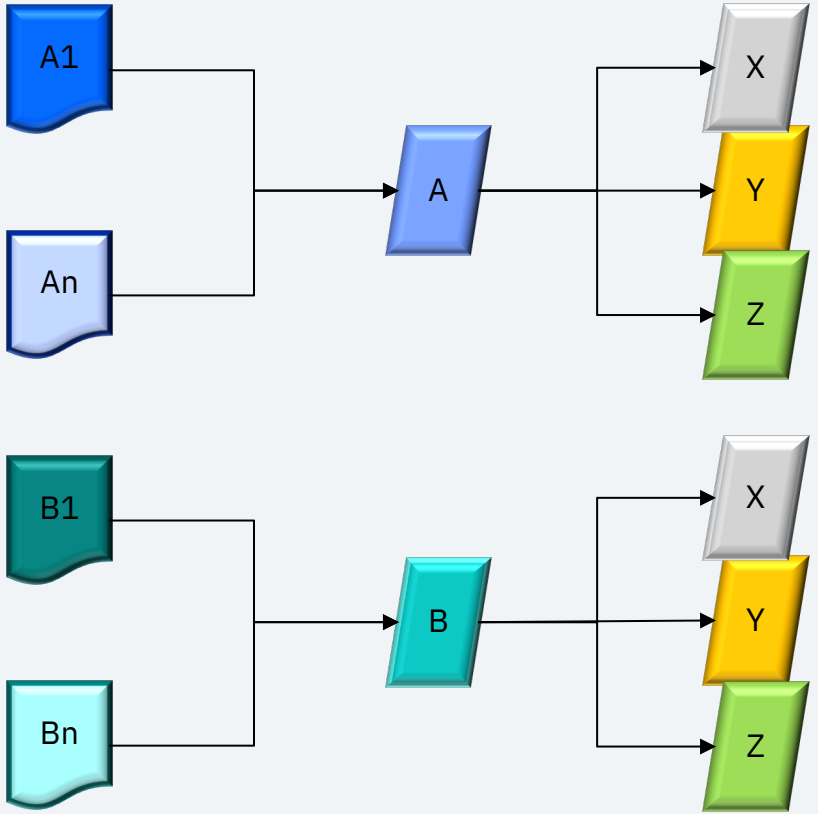
- Structure to structure mapping
- Flatten & unflatten
- Setting name-value pairs
- Ignore additional properties
- Conclusion
- What’s next?

Agenda

- **Structure to structure mapping**
- Flatten & unflatten
- Setting name-value pairs
- Ignore additional properties
- Conclusion
- What’s next?

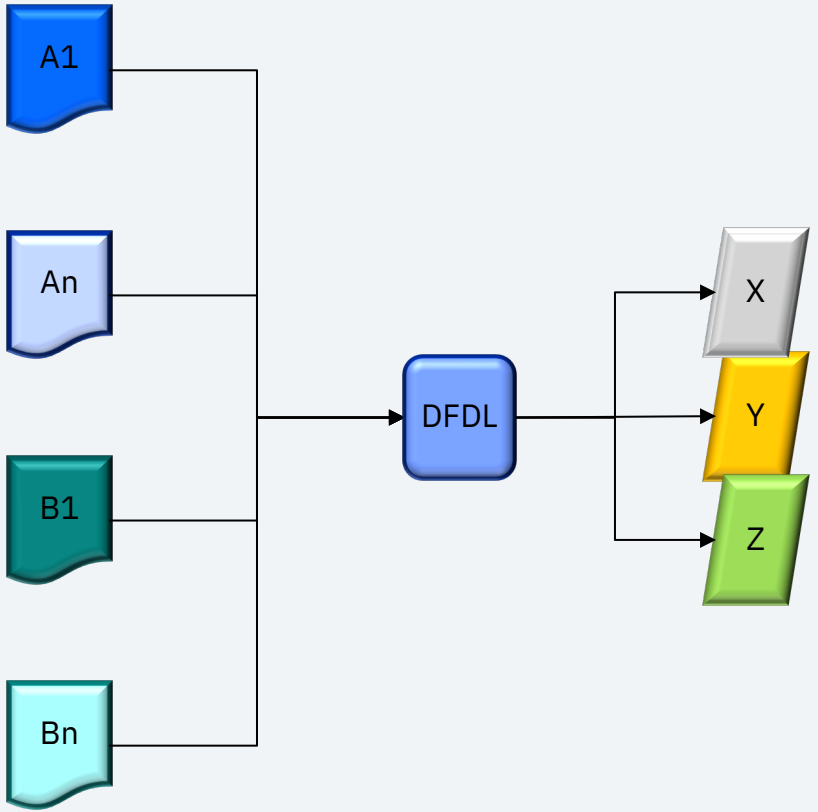
Problem Statement

Implementing REST services often requires multiple code updates to map between different structures which is time consuming and error prone.



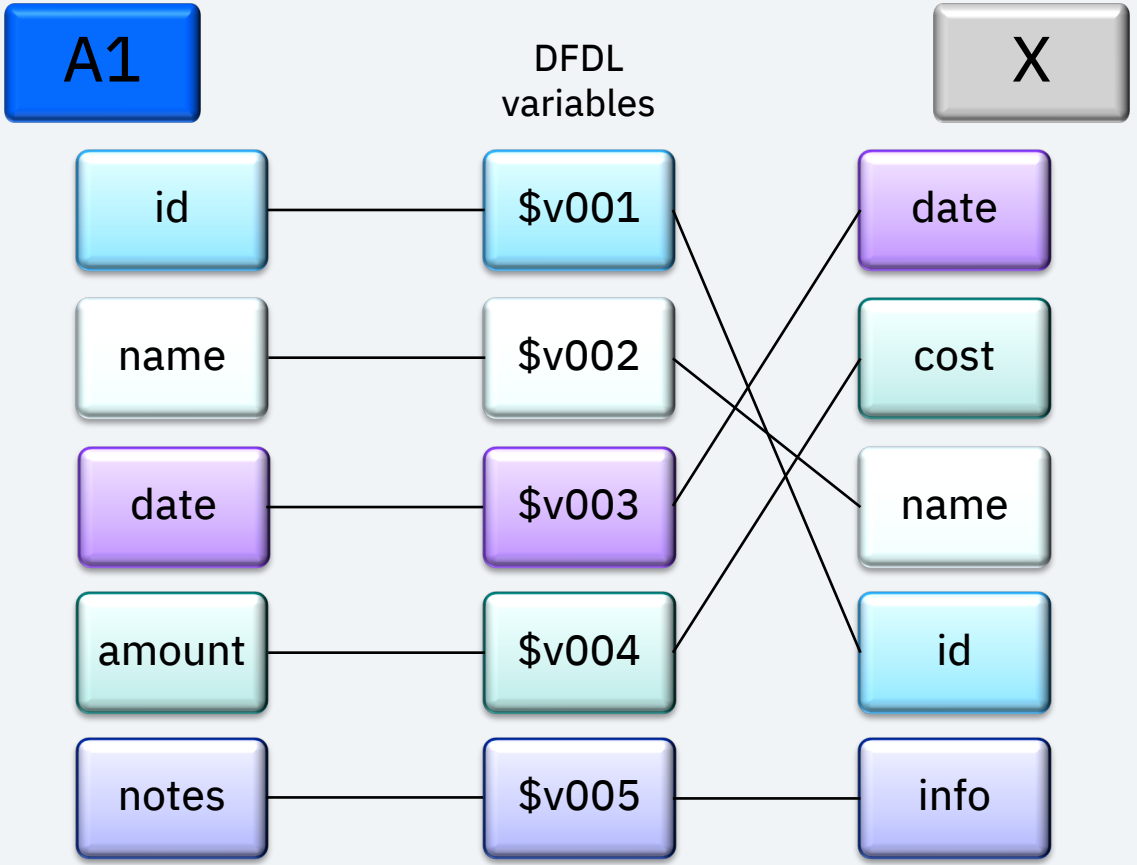
Value Statement

A mapping between multiple versions of dissimilar structures and a centralized structure can be maintained without requiring application changes.



Technical Details

Mapping dissimilar data can be accomplished through DFDL by setting and storing of external DFDL array variables.



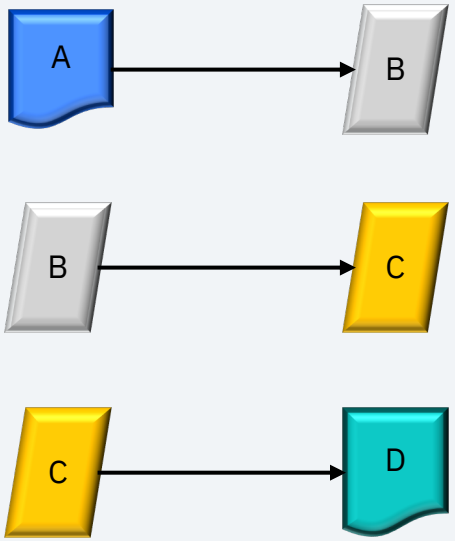
Technical Details

Structure to structure mapping can be performed in the following 3 ways:

document -> data

data -> data

data -> document



DFDL structure to structure: document -> data

```
{
  “BookingRequest”: {
    “Date”:”….”,
    “Flight”:”….”,
    “Origin”:”….”,
    “Destination”:”….”,
    “Passenger”:”….”,
    “PaymentInfo”:”….”
  }
}
```

LREC80 (Passenger Record)

LREC90 (Flight Record)

LRECA0 (Payment Record)

DFDL structure to structure: document -> data

```
try {

  tpf_dfdl_initialize_handle(&ih, “Input.dfdl.xsd”, “Input”, 0);

  tpf_dfdl_initialize_handle(&oh, “Output.dfdl.xsd”, “Output”, 0);

  // populate the DFDL variables from the document
  tpf_dfdl_serializeDoc(ih, docPtr, docLen, TPF_DFDL_JSON,
&outDataLen, NULL, 0);

  // make the DFDL variables accessible to the output
  tpf_dfdl_exportVariables(ih, oh);

  // create the output data
  outData = tpf_dfdl_createData(oh, 0);

}
```

Input (ns0):

```
<xs:element name="notes" type="xs:string" dfdl:length="16">
  <xs:annotation>
    <xs:appinfo source="http://www.ogf.org/dfdl/">
      <dfdl:setVariable ref="ns1:v001" value="{.}"/>
    </xs:appinfo>
  </xs:annotation>
</xs:element>
```

Output (ns1):

```
<xs:annotation>
  <xs:appinfo source="http://www.ogf.org/dfdl/">
    <dfdl:defineVariable name="v001" type="xs:string“
          external="true" tddt:array="true"/>
  </xs:appinfo>
</xs:annotation>

<xs:element name=“info" type="xs:string" dfdl:length="20“
dfdl:outputValueCalc="{$ns1:v001}"/>
```

DFDL structure to structure: data -> data

```
struct {
  struct {
    char *customer;
    int32_t order;
    char *notes;
    char *address;
    char *state;
    int32_t zip;
  } OrderInfo;
} purchaseRequest;
```

```
struct DFLREC {
  dft_siz DR26SIZ;
  dft_pky DR26KEY;
  struct {
    char DR26NAM[40];
    char DR26ADR[40];
    char DR26STA[2];
    unsigned int DR26ZIP;
    unsigned int DR26NUM;
    char DR26INF[1];
  } DR26K80;
};
```

DFDL structure to structure: data -> data

```
try {

 tpf_dfdl_initialize_handle(&ih, “Input.dfdl.xsd”, “Input”, 0);

 tpf_dfdl_initialize_handle(&oh, “Output.dfdl.xsd”, “Output”, 0);

 // populate the DFDL variables from the data
 tpf_dfdl_setData(ih, pData, dataSize);
 tpf_dfdl_readData(ih, 0);

 // make the DFDL variables accessible to the output
 tpf_dfdl_exportVariables(ih, oh);

 // create the output data
 outData = tpf_dfdl_createData(oh, 0);

}
```

Input (ns0):

```
<xs:element name="customer" type="xs:string"
dfdl:lengthKind="delimited" tddt:indirectKind="pointer"
tddt:indirectLength="8" >
  <xs:annotation>
    <xs:appinfo source="http://www.ogf.org/dfdl/">
      <dfdl:setVariable ref="ns1:v001" value="{.}"/>
    </xs:appinfo>
  </xs:annotation>
</xs:element>
```

Output (ns1):

```
<xs:annotation>
  <xs:appinfo source="http://www.ogf.org/dfdl/">
    <dfdl:defineVariable name="v001" type="xs:string“
          external="true" tddt:array="true"/>
  </xs:appinfo>
</xs:annotation>

<xs:element name="DR26NAM" type="xs:string"
dfdl:length="40" dfdl:outputValueCalc="{$ns1:v001}"/>
```

DFDL structure to structure: data -> document

LREC80 (Passenger Record)

LREC90 (Flight Record)

LRECA0 (Payment Record)

```
{
  “BookingResponse”: {
    “Date”:”....”,
    “Flight”:”....”,
    “Origin”:”....”,
    “Destination”:”....”,
    “Passenger”:”....”,
    “PaymentInfo”:”....”
  }
}
```

DFDL structure to structure: data -> document

```
try {

 tpf_dfdl_initialize_handle(&ih, “Input.dfdl.xsd”, “Input”, 0);

 tpf_dfdl_initialize_handle(&oh, “Output.dfdl.xsd”, “Output”, 0);

 // populate the DFDL variables from the data
 tpf_dfdl_setData(ih, pData, dataSize);
 tpf_dfdl_readData(ih, 0);

 // make the DFDL variables accessible to the output
 tpf_dfdl_exportVariables(ih, oh);

 // create the output document
 docPtr = tpf_dfdl_buildDoc(oh, &docLen,
TPF_DFDL_JSON, 0);

}
```

Input (ns0):

```
<xs:annotation>
  <xs:appinfo source="http://www.ogf.org/dfdl/">
    <dfdl:defineVariable name="v001" type="xs:string“
          external="true" tddt:array="true"/>
  </xs:appinfo>
</xs:annotation>

 <xs:element name=“info" type="xs:string" dfdl:length=“20">
  <xs:annotation>
    <xs:appinfo source="http://www.ogf.org/dfdl/">
      <dfdl:setVariable ref="ns0:v001" value="{.}"/>
    </xs:appinfo>
  </xs:annotation>
</xs:element>
```

Output (ns1):

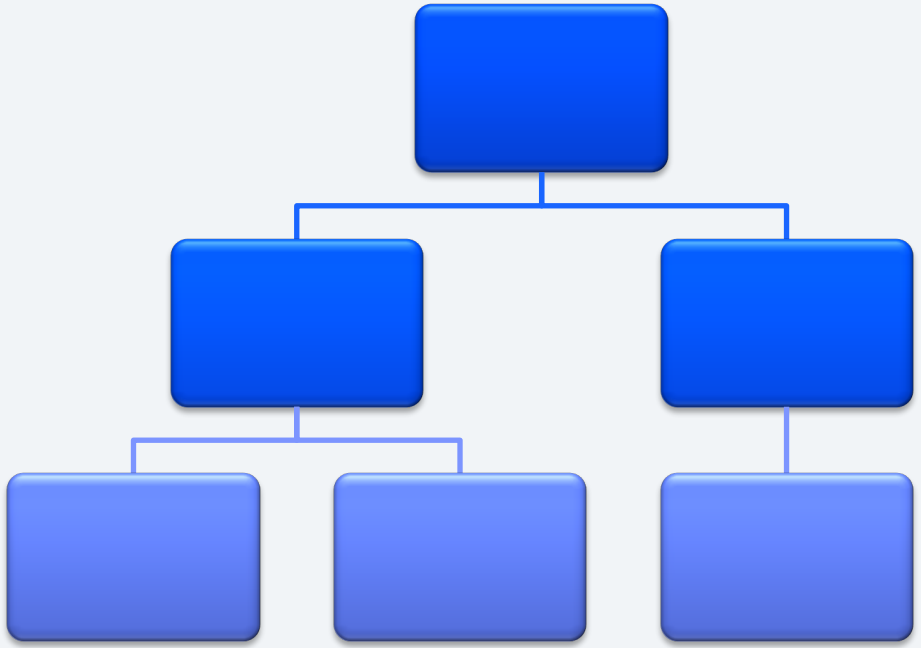
```
<xs:element name=“notes" type="xs:string"
dfdl:inputValueCalc="{$ns0:v001}"/>
```

Agenda

- Structure to structure mapping
- **Flatten & unflatten**
- Setting name-value pairs
- Ignore additional properties
- Conclusion
- What’s next?

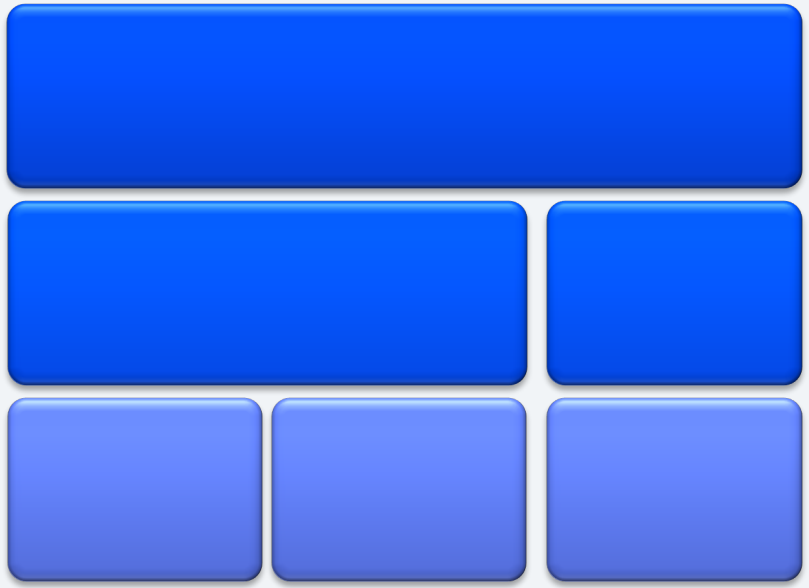
Problem Statement

z/TPF REST generated artifacts use pointers for variables size data which makes it difficult to copy the data elsewhere or file it down for asynchronous processing.



Value Statement

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



Technical Details

PJ45994 (June 2020) - tpfdfdlgen pointer support created the ability to generate DFDL for pointers defined within C structures.

The **tpf_dfdl_createData** function can be used to create a contiguous blob with all pointers changed to offsets.

The **tpf_dfdl_readData** function can be used to convert all offsets within a contiguous blob to pointers.

Flatten Example:

```
// Convert data to contiguous blob
try {
    tpf_dfdl_initialize_handle(&dh, REQ_DFDL, REQ_ROOT, 0);
    tpf_dfdl_setData(dh, inData, dataLen);
    blob = tpf_dfdl_createData(dh, &blobLen, TPF_DFDL_PTR2OFF);
}  catch (std::exception &e) {
    // handle error for e.what()
}
tpf_dfdl_terminate_handle(dh);
```

Flatten Example: Before

```
17791300- 00000004 00000000 00000000 17791400 ........ .....`..
17791310- 00000000 1778c3c0 00001388 00000000 ......C{ ...h....
17791320- 00000000 1778c400 00003139 0000003f ......D. ........
17791330- 00000002 00000000 00000000 17791800 ........ .....`..
17791340- 00000000 1778c640 00016122 00000000 ......F  ../.....
17791350- 00000000 1778c680 00000000 00000019 ......F. ........

17791400- 00000005 0000000a 0000000f 00000014 ........ ........

1778c3c0- e29489a3 8800                       Smith.

1778c400- d5e800                              NY.

...
```

Flatten Example: After

```
17798000- 00000004 00000000 00000000 00000060 ........ .......-
17798010- 00000000 00000070 00001388 00000000 ........ ...h....
17798020- 00000000 00000076 00003139 0000003f ........ ........
17798030- 00000002 00000000 00000000 00000079 ........ .......`
17798040- 00000000 000000ca 00016122 00000000 ........ ../.....
17798050- 00000000 000000d0 00000000 00000019 .......} ........
17798060- 00000005 0000000a 0000000f 00000014 ........ ........
17798070- e29489a3 8800d5e8 00000000 00000000 Smith.NY ........
17798080- b9000027 10000000 00000000 00000000 ........ ........
17798090- bf000000 00000000 1e000000 00000000 ........ ........
177980a0- c2000001 00000000 00000000 00000000 B....... ........
177980b0- c7000000 00000000 00d19695 85a200c1 G....... .Jones.A
177980c0- e900c889 939300c7 c100c481 a589a200 Z.Hill.G A.Davis.
177980d0- c3c100                                CA.
```

Unflatten Example:

```
// Convert contiguous blob to pointer form
try {
    tpf_dfdl_initialize_handle(&dh, REQ_DFDL, REQ_ROOT, 0);
    tpf_dfdl_setData(dh, blob, blobLen);
    tpf_dfdl_readData(dh, TPF_DFDL_OFF2PTR);
}  catch (std::exception &e) {
    // handle error for e.what()
}
tpf_dfdl_terminate_handle(dh);
```

Unflatten Example: Before

```
17798000- 00000004 00000000 00000000 00000060 ........ .......-
17798010- 00000000 00000070 00001388 00000000 ........ ...h....
17798020- 00000000 00000076 00003139 0000003f ........ ........
17798030- 00000002 00000000 00000000 00000079 ........ .......`
17798040- 00000000 000000ca 00016122 00000000 ........ ../.....
17798050- 00000000 000000d0 00000000 00000019 .......} ........
17798060- 00000005 0000000a 0000000f 00000014 ........ ........
17798070- e29489a3 8800d5e8 00000000 00000000 Smith.NY ........
17798080- b9000027 10000000 00000000 00000000 ........ ........
17798090- bf000000 00000000 1e000000 00000000 ........ ........
177980a0- c2000001 00000000 00000000 00000000 B....... ........
177980b0- c7000000 00000000 00d19695 85a200c1 G....... .Jones.A
177980c0- e900c889 939300c7 c100c481 a589a200 Z.Hill.G A.Davis.
177980d0- c3c100                                CA.
```

Unflatten Example: After

```
17798000- 00000004 00000000 00000000 17798060 ........ .....`.-
17798010- 00000000 17798070 00001388 00000000 .....`.. ...h....
17798020- 00000000 17798076 00003139 0000003f .....`.. ........
17798030- 00000002 00000000 00000000 17798079 ........ .....`.`
17798040- 00000000 177980ca 00016122 00000000 .....`.. ../.....
17798050- 00000000 177980d0 00000000 00000019 .....`.} ........
17798060- 00000005 0000000a 0000000f 00000014 ........ ........
17798070- e29489a3 8800d5e8 00000000 00177980 Smith.NY ......`.
17798080- b9000027 10000000 00000000 00177980 ........ ......`.
17798090- bf000000 00000000 1e000000 00177980 ........ ......`.
177980a0- c2000001 00000000 00000000 00177980 B....... ......`.
177980b0- c7000000 00000000 00d19695 85a200c1 G....... .Jones.A
177980c0- e900c889 939300c7 c100c481 a589a200 Z.Hill.G A.Davis.
177980d0- c3c100                                CA.
```

Agenda

- Structure to structure mapping
- Flatten & unflatten
- **Setting name-value pairs**
- Ignore additional properties
- Conclusion
- What’s next?

Value Statement

Name-value pairs may be set as part of a REST interface, requiring no application updates.

Setting name-value pairs
REST example

POST /travel/booking/v1
apiKey: 172H527JE9S52VNU
Content-Length: 651
Content-Type: application/json

```
{
  “BookingRequest”: {
    “Date”:”….”,
    “Flight”:”….”,
    “Origin”:”….”,
    “Destination”:”….”,
    “Passenger”:”….”,
    “PaymentInfo”:”….”
  }
}
```

Name-value pairs:

ISrvcName: “trBookv1”
ISrvcVersion: “1.0.0”
Requestor: “172H527JE9S52VNU”

Technical Details

PJ45427 (Oct 2018) – Added the ability to reference name-value pairs from DFDL as a DFDL variable.

The **setVariable** DFDL annotation can be used to set the value of any defined name-value pair.

User defined name-value pairs can be defined in **tpfNameValue.user.dfdl.xsd**; while system defined name-value pairs remain in tpfNameValue.lib.dfdl.xsd.

User defined name-value pair: defining

tpfNameValue.user.dfdl.xsd:

```
<xs:annotation>
        <xs:appinfo source="http://www.ogf.org/dfdl/">
                <dfdl:defineVariable name=“Requestor" type="xs:string" external="true" />
        </xs:appinfo>
</xs:annotation>
```

User defined name-value pair: setting

Request.gen.dfdl.xsd:

xmlns:nv="http://www.ibm.com/xmlns/prod/ztpf/name-value"

```
<xs:import namespace="http://www.ibm.com/xmlns/prod/ztpf/name-value"
      schemaLocation="tpfNameValue.lib.dfdl.xsd"/>

<xs:element name=“apiKey" type="xs:string" dfdl:length="16">
            <xs:annotation>
                  <xs:appinfo source="http://www.ogf.org/dfdl/">
                        <dfdl:setVariable ref="nv:Requestor" value="{.}"/>
                  </xs:appinfo>
            </xs:annotation>
</xs:element>
```

Agenda

- Structure to structure mapping
- Flatten & unflatten
- Setting name-value pairs
- **Ignore additional properties**
- Conclusion
- What’s next?

Problem Statement

Unlike REST provider, the DFDL serialize functions do not provide a way to ignore additional properties/elements within the JSON/XML document that are not defined within the DFDL schema.

Value Statement

The tpf_dfdl_serializeData function can be used to efficiently retrieve only the properties/elements from an XML/JSON document that are of interest.

Technical Details

The tpf_dfdl_serializeData function doesn’t require any particular order of elements in the document.

A new option of TPF_DFDL_IGNORE will allow elements to exist in the document that are not the DFDL schema.
- tpf_dfdl_serializeData(dh, xh, root_name, TPF_DFDL_IGNORE)

Agenda

- Structure to structure mapping
- Flatten & unflatten
- Setting name-value pairs
- Ignore additional properties
- **Conclusion**
- What’s next?

Conclusion: PJ46213
April 2021

- A mapping between multiple versions of dissimilar structures and a centralized structure can be maintained without requiring application changes.
- DFDL can transform structures with pointers to a flattened representation with offsets efficiently and in a standardized way.
- Name-value pairs may be set as part of a REST interface, requiring no application updates.
- The tpf_dfdl_serializeData function can be used to efficiently retrieve only the properties/elements from an XML/JSON document that are of interest.

What’s next?

Future possibilities:

- Allow customization for float/double document format
- DFDL display command to help with diagnosis and validation
- DFDL offline validation and library support (run the z/TPF DFDL parser on linux on Z).

Thank you

Bradd Kadlecik
z/TPF Development
—
braddk@us.ibm.com

Virtual TPFUG Q&A

Summary of Q&A from the virtual TPFUG event:

Question	Answer
Can you access DFDL variables programatically?	Yes, you can use tpf_dfdl_getVariable or tpf_dfdl_setVariable to programatically work with DFDL variables.