REST Structure to Structure Mapping Support

Web Services Subcommittee

Bradd Kadlecik

2024 TPF Users Group Conference
May 5-8, New Orleans, LA

IBM Z

IBM

Problem Statement

To map dissimilar data between the REST interface and the application data format, the data must undergo 2 transformations. One transformation between the REST interface and a compatible data format. Another transformation between the REST compatible data format and the application data format.

Users

Zach
Application developer

I need to update my application to support a REST interface.

As-Is User Story

Zach uses an OpenAPI document to generate the C structures and DFDL schemas for the request and reply formats using tpfrestgen.

Generated C structure:

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

As-Is User Story

Zach needs to be able to map the data with pointers to an LREC format that his application expects involving no pointers.

```
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;
};
```

As-Is User Story

Zach creates a DFDL structure-to-structure mapping for a data-to-data transformation.

REST format:

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

DFDL variables:

$DR26NAM

$DR26ADR

$DR26STA

$DR26ZIP

$DR26NUM

$DR26INF

$DR26SIZ = valueLength(notes) + 93

Application format:

```
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;
};
```

Pain Points

- A REST compatible data format must be maintained for each REST interface resulting in additional artifacts.
- The application must be updated to perform DFDL data-to-data mappings for both the request and response.
- Additional CPU overhead for having to perform 2 DFDL transformations.

Value Statement

An application developer can reduce the amount of maintenance and processing required to map various REST requests and responses to an internal application superstructure.

To-Be User Story

Zach uses an OpenAPI document to generate the C structures and DFDL schemas for the request and reply formats using tpfrestgen.

Generated C structure:

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

To-Be User Story

Zach needs to be able to map the data with pointers to an LREC format that his application expects involving no pointers. By using REST structure-to-structure mapping, Zach can ignore the generated C structure and instead create a mapping for the LREC format his application expects.

```
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;
};
```

To-Be User Story

Zach creates a DFDL structure-to-structure mapping for a **document-to-data** transformation to convert the request directly to the desired format.

REST format:

```
“purchaseRequest”:{
  “OrderInfo”:{
    “customer” : ...,
    “order”: ...,
    “notes”: ...,
    “address”: ...,
    “state”: ...,
    “zip”: ...
  }
}
```

DFDL variables:

$DR26NAM

$DR26ADR

$DR26STA

$DR26ZIP

$DR26NUM

$DR26INF

$DR26SIZ = valueLength(notes) + 93

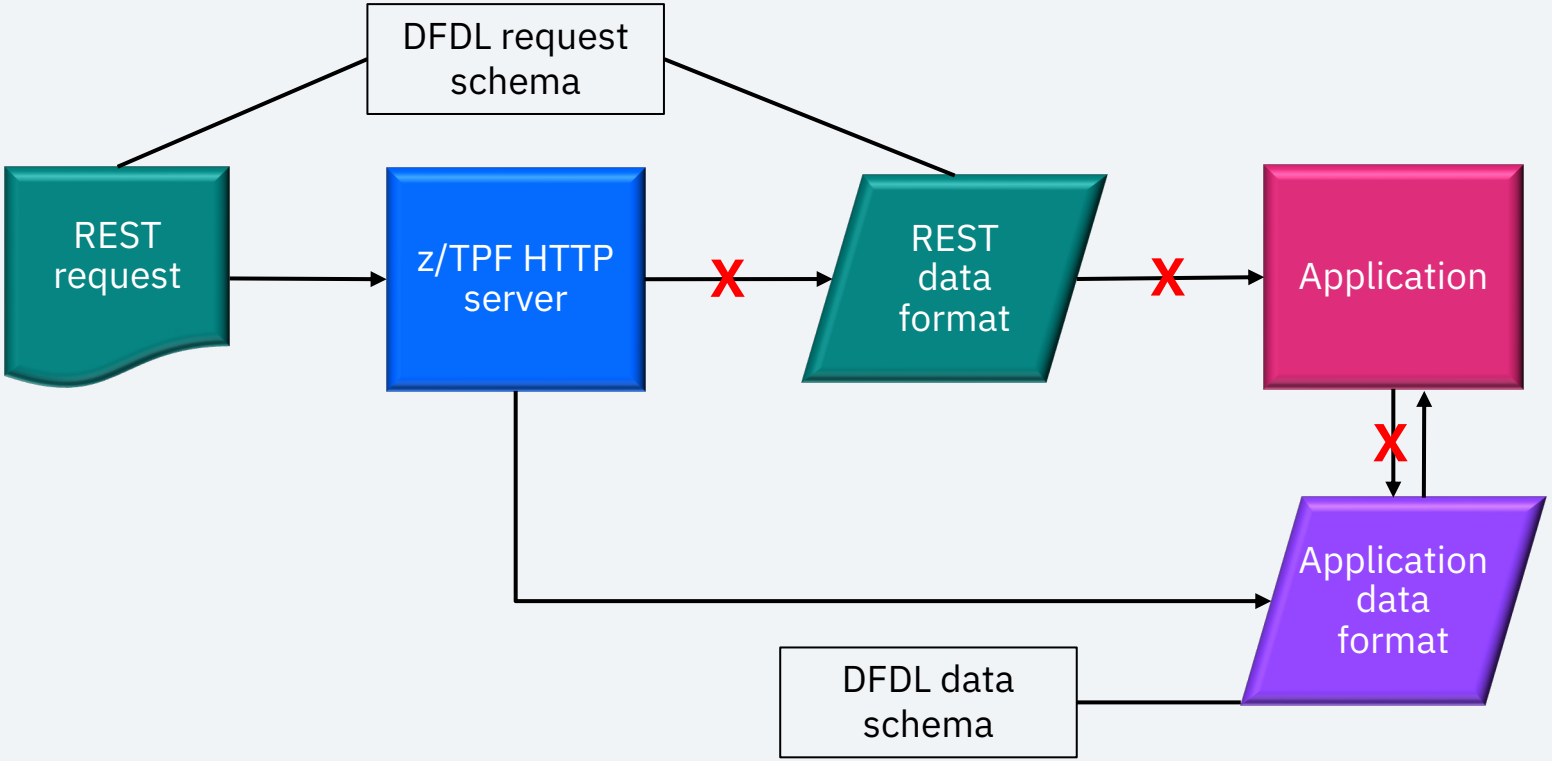
Application format:

```
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;
};
```

To-Be User Story

Zach does not need to add any additional code to his application to map the REST request to the application data format. By defining the data object in the z/TPF service descriptor, the data interface for REST matches his application interface.

Technical Details – PJ47073 (Aug 2023)
Overview



Technical Details – PJ47073 (Aug 2023)
REST structure to structure mapping support

Dissimilar data structures are defined through the *data* object in the z/TPF service descriptor.

External DFDL variables are used to map data between the REST interface and the dissimilar data structure.

The source DFDL schema uses setVariable to populate data values that are to be mapped.

The target DFDL schema uses calculated fields to store a data value for any given element.

Technical Details – PJ47073 (Aug 2023)
z/TPF service descriptor example

```
"operationId": "s2sComplexReq",
"request": {
   "schema":
"s2sComplexReqRequest_t.gen.dfdl.xsd",
   "root": "s2sComplexReqRequest_t",
   "data":{
      "DFDLValidation": true,
      "schema":"s2sData.drvr.dfdl.xsd",
      "root" : "s2sData"
   }
},
```

```
"response": {
   "schema":
"s2sComplexReq_200Response_t.gen.dfdl.xsd",
   "root": "s2sComplexReq_200Response_t",
   "status": [200],
   "data":{
      "schema":"s2sData.drvr.dfdl.xsd",
      "root" : "s2sData"
   }
}
```

Technical Details – PJ47073 (Aug 2023)
DFDL schema example - request

s2sComplexReqRequest_t.gen.dfdl.xsd :

```
xmlns:s2sd=http://www.ibm.com/xmlns/prod/ztpf/dfdl/drvr/s2sData

<xs:import
namespace="http://www.ibm.com/xmlns/prod/ztpf/dfdl/drvr/s2sData"  schemaLocation="s2sData.drvr.dfdl.xsd"/>

<xs:element name="connection" type="xs:string"
tddt:indirectKind="pointer" tddt:indirectLength="8"
dfdl:lengthKind="delimited" dfdl:lengthUnits="bytes"
dfdl:terminator="%NUL;" dfdl:textTrimKind="none"
default="">
   <xs:annotation>
      <xs:appinfo source="http://www.ogf.org/dfdl/">
         <dfdl:setVariable ref="s2sd:drsricc" value="{.}" />
      </xs:appinfo>
   </xs:annotation>
</xs:element>
```

s2sData.drvr.dfdl.xsd :

```
<dfdl:defineVariable name="drsricc" type="xs:string"
external="true" tddt:array="true" defaultValue=""/>

<xs:element name= "dr40rsricc" type="xs:string"
dfdl:outputValueCalc="{$drsricc}" tddt:indirectKind="pointer"
tddt:indirectLength="8" dfdl:lengthKind="delimited"
dfdl:lengthUnits="bytes" dfdl:terminator="%NUL;"
dfdl:textTrimKind="none"/>
```

Technical Details – PJ47073 (Aug 2023)
DFDL schema example - response

s2sComplexReq_200Response_t.gen.dfdl.xsd :

xmlns:**s2sd**=http://www.ibm.com/xmlns/prod/ztpf/dfdl/drvr/s2sData

<xs:import namespace="http://www.ibm.com/xmlns/prod/ztpf/dfdl/drvr/s2sData" schemaLocation="s2sData.drvr.dfdl.xsd"/>

<xs:element name="**connection**" type="xs:string" dfdl:inputValueCalc="{$**s2sd**:rsricc}"/>

s2sData.drvr.dfdl.xsd :

<dfdl:defineVariable name="**rsricc**" type="xs:string" external="true" tddt:array="true" defaultValue=""/>

<xs:element name= "**dr40rsricc**" type="xs:string" dfdl:outputValueCalc="{$drsricc}" tddt:indirectKind="pointer" tddt:indirectLength="8" dfdl:lengthKind="delimited" dfdl:lengthUnits="bytes" dfdl:terminator="%NUL;" dfdl:textTrimKind="none">
  <xs:annotation>
    <xs:appinfo source="http://www.ogf.org/dfdl/">
      <dfdl:setVariable ref="rsricc" value="{.}" />
    </xs:appinfo>
  </xs:annotation>
</xs:element>

Technical Details – PJ47073 (Aug 2023)
REST structure to structure exclusion

Elements that are not required can be excluded from the REST generated output (created by DFDL) if the value matches the default.

<xs:element name="connection" type="xs:string"
dfdl:inputValueCalc="{$s2sd:rsricc}" ~~default=""~~/>

DFDL specification: “It is a Schema Definition Error if [inputValueCalc] is specified on an element which has an XSD fixed or default property.”

To allow for exclusion of an element containing inputValueCalc, a z/TPF unique attribute of “default” can be used.

<xs:element name="connection" type="xs:string"
dfdl:inputValueCalc="{$s2sd:rsricc}" tddt:default="" />

As-Is User Story

After updating the DFDL schema files to do the structure-to-structure mapping, Zach needs to write some code to test out his mappings on z/TPF and keep reloading his DFDL schemas for corrections needed.

Pain Points

- There is no way to easily test out the DFDL structure to structure mappings.

Value Statement

DFDL structure to structure mappings can be tested and validated on Linux for IBM Z.

To-Be User Story

Zach tests out his mappings offline using tpfdatamap on Linux for IBM Z and can quickly modify his DFDL schemas for corrections needed.

Technical Details – PJ47073 (Aug 2023)
tpfdatamap transform on Linux for IBM Z

tpfdatamap **transform** -s source -t target -i infile -o outfile [-d]
source_filename target_filename

- Both input and output files can be either a document or data format.
- Supports data-to-document, data-to-data, document-to-data, and document-to-document transformations.
- Supported document formats: BSON (.bson), CSV (.csv), JSON (.json), Java properties (.properties), and XML (.xml).
- See *man tpfdatamap* for more information.

Technical Details – PJ47073 (Aug 2023)
tpfdatamap transform example

Document-to-data transformation:

> tpfdatamap t -s body -t s2sData -i s2sComplexReq.json -o s2sData.bin s2sComplexReqRequest_t.gen.dfdl.xsd s2sData.drvr.dfdl.xsd

DATAMAP0001I Processing completed

Data-to-document transformation:

> tpfdatamap t -s s2sData -t body -i s2sData.bin -o s2sComplexResp.json s2sData.drvr.dfdl.xsd s2sComplexReq_200Response_t.gen.dfdl.xsd

DATAMAP0001I Processing completed

Conclusion

PJ47073 (Aug 2023): REST structure to structure mapping support

- An application developer can reduce the amount of maintenance required to map various REST requests and responses to an internal application superstructure.
- The mapping of various REST requests and responses to an internal application superstructure can be done with 1 DFDL transformation instead of 2, thereby reducing CPU consumption.
- DFDL structure to structure mappings can be tested and validated on Linux for IBM Z.

Thank you