REST Enhancements

SOA Subcommittee

—

Bradd Kadlecik
z/TPF Development

Agenda:

REST Consumer (PJ45005)

Unordered JSON (PJ45191)

Potential REST updates

Agenda:

REST Consumer (PJ45005)

Unordered JSON (PJ45191)

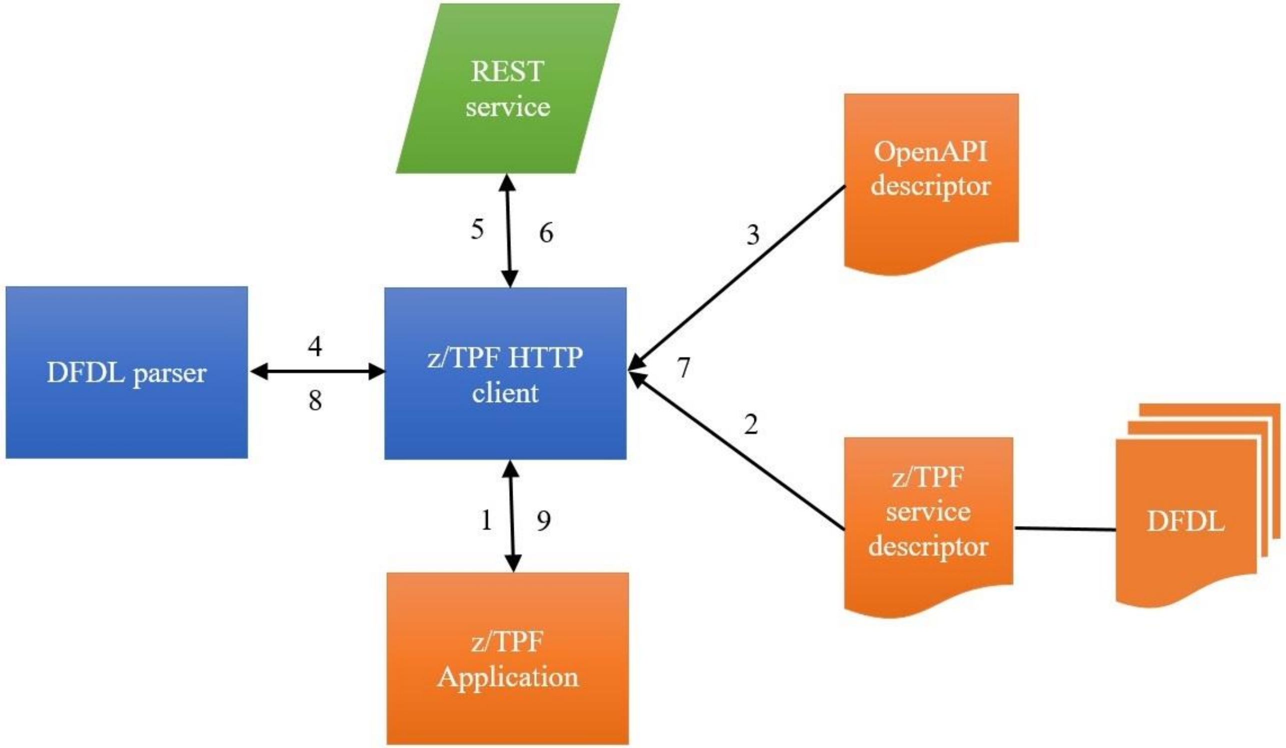
Potential REST updates

REST Consumer (PUT14 - PJ45005)

Artifacts:
OpenAPI (swagger)
Service descriptor
DFDL

Infrastructure:
Enhanced HTTP client
High speed connector

APIs:
Same as calling Java on z/TPF

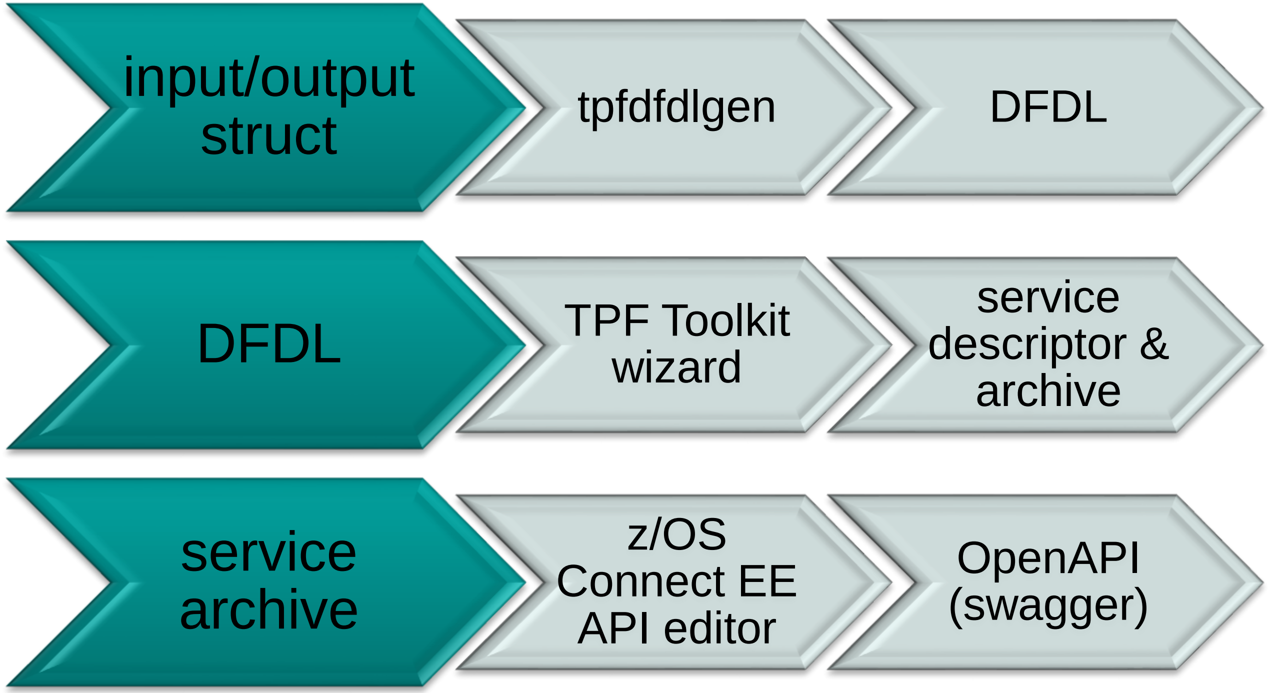


REST Consumer (API Creation)

Same process as for REST producer.

Need to create request/reply structures.

The service archive (.sar) is only needed for z/OS Connect EE API editor to create the OpenAPI descriptor. It is not used on z/TPF.



REST Consumer (z/TPF service descriptor)

```
{
  “version”:1,
  “operationId”: “RESTapiCall”,
  “request”: {
     “schema”: “apiReq_parms.gen.dfdl.xsd”,
     “root”: “apiReq_parms”
  },
  “response”: {
     “schema”: “apiResp_parms.gen.dfdl.xsd”,
     “root”: “apiResp_parms”
  },
  “timeout”: 2000
}
```

The operationId must match an operationId in a OpenAPI descriptor.

The operationId is what is referenced by the API.

The “providerType” and “providerName” properties are optional.

Destination (host) information is retrieved from the OpenAPI descriptor.

REST Consumer (API example)

Simplified coding interface:

Greatly reduces the amount of code needed to call a RESTful service. (100s LOC -> 1 function call)

less code = less bugs

```
#include <tpf/services.h>

struct apiReq_parms reqData;
struct tpf_srvc_resp *srvcResp = NULL;
int reqLen = sizeof(reqData);
int rc;

rc = tpf_srvcInvoke(“RESTapiCall”, &reqData,
                                        reqLen, &srvcResp, 0);
if (rc != TPF_SRVC_INVOKE_SUCCESS) {
  // Handle system error (srvcResp == NULL)
}
else if (srvcResp->status != TPF_SRVC_OK) {
  // Handle application error
}
```

REST Consumer (API examples)

Synchronous call

```
tpf_srvcInvoke("RESTapiCall", reqData, reqLen,
                              &srvcResp, 0);
```

- or -

```
tpf_srvcInvoke_ext("RESTapiCall", srvcReq,
                                  &srvcResp, NULL, 0);
```

Asynchronous call

```
tpf_srvcInvoke_ext("RESTapiCall", srvcReq,
                                  &srvcResp, asyncParms, 0);
```

REST Consumer (OpenAPI descriptor)

The “host” property contains the host name (or IP) with optional port specified.

Example of local Java service on z/TPF:
“host”:”localhost”

Example of remote service with non-persistent sessions:
“host”:”www.ibm.com:80”

Example of remote service with persistent sessions:
“host”:”www.ibm.com”

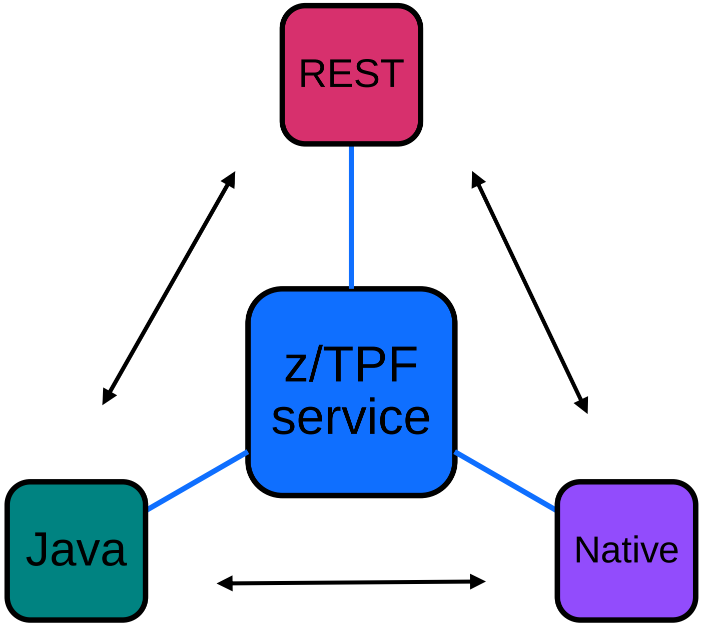
Endpoint group descriptor:
<aliasHostname>www.ibm.com</aliasHostname>

The OpenAPI descriptor controls whether the call is local or remote.

A matching <aliasHostname> in an endpoint group descriptor controls whether the session is persistent or non-persistent.

Switching all callers from calling Java off platform to calling Java on platform would only require changing this host property.

z/TPF service routing (PUT14)



REST call -> Native service (PJ44281)

REST call -> Java service (PJ43892)

Native call -> Java service (PJ43892)

Java call -> RESTful service (PJ43892)

Java call -> Native service (PJ44844)

Native call -> RESTful service (PJ45005)

REST Consumer (PUT14 - PJ45005)

Simpler:

API interface deals with input/output structures rather than building XML/JSON, HTTP headers, etc.

Configurable:

The host name for the REST service is kept in the OpenAPI descriptor making it easier to configure between test and production systems.

Integrated:

Uses the same API as calling Java on z/TPF so having the REST service on or off z/TPF requires no code change.

Makes use of High Speed Connector support for configuring persistent connections.

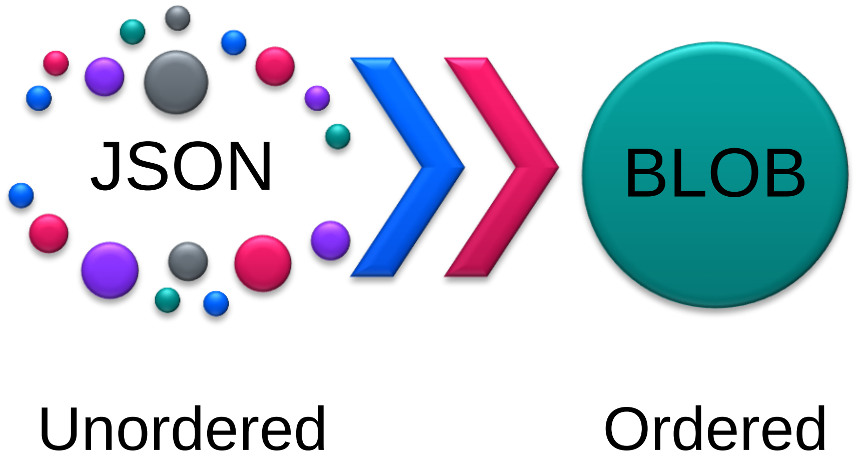
Agenda:

REST Consumer (PJ45005)

Unordered JSON (PJ45191)

Potential REST updates (2018)

Unordered JSON (PUT15 - PJ45191)



JSON does not dictate an order of elements.

z/TPF service descriptor was updated to indicate if unordered JSON is allowed.

Serializing unordered elements will be less efficient than ordered.

Unordered JSON (z/TPF service descriptor)

Default behavior requires the order of JSON elements/properties to match the order in DFDL.

Setting “unordered” to true in the service descriptor allows any order for the JSON or XML in the HTTP request or response body.

Java interfaces use JSON but maintain order.

```
{
  "version":1,
  "operationId":"loaderUpdate",
  "description": "This is a service Descriptor",
  "providerType":"Program",
  "provider":"CNG0",
  “unordered”:true,
  "request":{
      "schema":"isetinfo.gen.dfdl.xsd",
      "root":"isetinfo“
      },
    "response": {
        "schema": "lsetinfo.gen.dfdl.xsd",
        "root": "lsetinfo“
    },
  "timeout":5000
}
```

Agenda:

REST Consumer (PJ45005)

Unordered JSON (PJ45191)

Potential REST 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.

OpenAPI Path templating (URL parameters)

Path templating in OpenAPI is using the curly braces {} to parameterize some portion of the URL path.

Example:

PUT /tpf/service/loadest/appltest?action=dea

```
"/loadset/{lsname}":{
  "put":{
    "operationId":"tpfModLset",
    "parameters":[
    {
      "name":"lsname",
      "in":"path",
      "required":true,
      "type":"string"
    },
    {
      "name":"action",
      "in":"query",
      "required":true,
      "type":"string"
    }]
```

OpenAPI XML Objects (allows XML attributes)

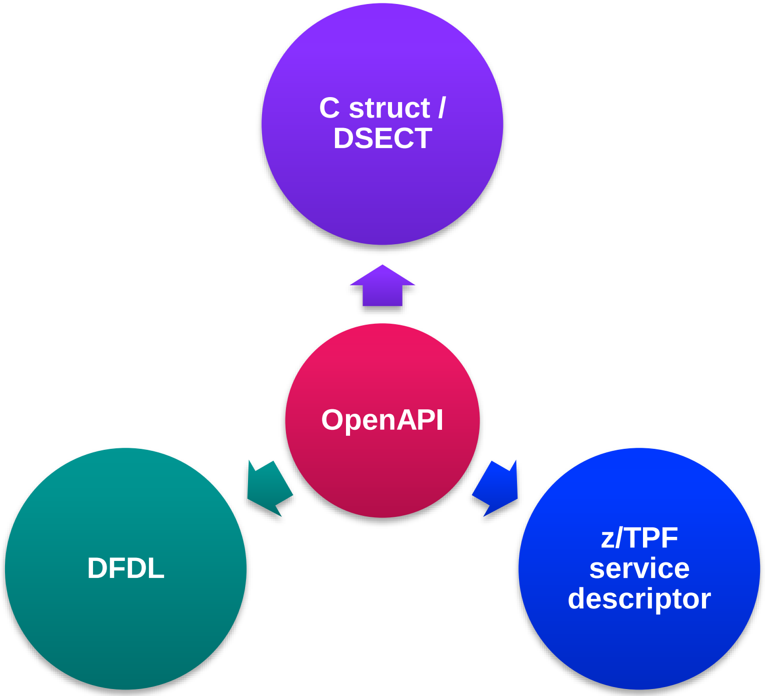
XML Objects are used for XML only information.

XML attribute: “stdbid”

```
<?xml version=“1.0” encoding=“utf-8”?>
<stdhd stdbid=“BD”>
   <stdchk>1</stdchk>
   <stdctl>0</stdctl>
   <stdpgm>ABCD</stdpgm>
   <stdfch>0</stdfch>
   <stdbch>0</stdbch>
</stdhd>
```

```
C2C40100C1C2C3C4
0000000000000000
```

Swagger codegen for z/TPF (Generate artifacts for REST)



Development process:

1. Use an existing OpenAPI document.
2. Generate C structures, assembler DSECTs, DFDL, and z/TPF service descriptors to use on z/TPF.
3. Code the interface using the the generated C structures or assembler DSECTs for request/response data.

Thank You!

Questions or Comments?

Trademarks