REST Provider Support (PJ44281)

SOA Subcommittee

Bradd Kadlecik
z/TPF Development

IBM **z/TPF**
April 4th, 20017

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

- Overview
- API Creation
- Artifacts
- REST services
- What’s Next

REST current support

- Expensive to code to handle all the parsing required
- No standardized way of describing REST services
- No infrastructure for providing API discovery services

REST provider support

- Tooling from TPF Toolkit and z/OS Connect provides simplified way of creating REST services without much code
- OpenAPI (aka Swagger) is used for REST implementation
- API Discovery for API Connect, Swagger editor, etc

REST current support

Create a service wrapper program that:
1) takes a HTTP request structure as input
2) parses the XML or JSON
3) puts the data where needed to call the service

```
void <PROG> (tpf_httpsvr_req *request, tpf_httpsvr_token tok);
```

REST provider support

Create a service wrapper program that:
1) takes a single structure for input
- no HTTP, XML, JSON knowledge
2) puts the data where needed to call the service

```
void <PROG> (void *input, unsigned int len, tpf_srvc_token tok);
```

REST current support

Create response handlers that:
1) build a HTTP response structure
2) create the XML or JSON based on “Accept” request header.

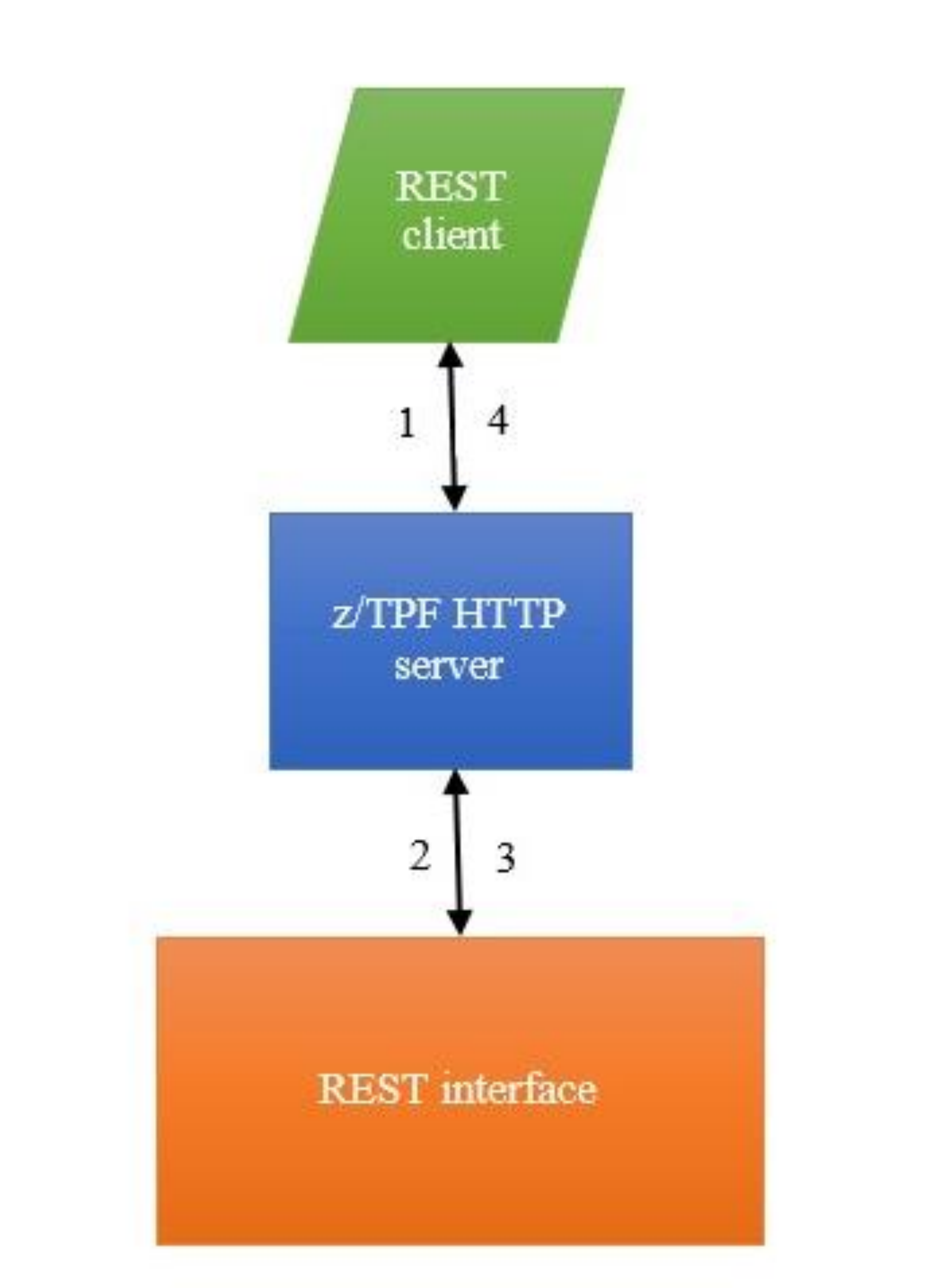
```
int tpf_httpSendResponse(tpf_httpsvr_token token,
                         tpf_httpsvr_resp *response);
```

REST provider support

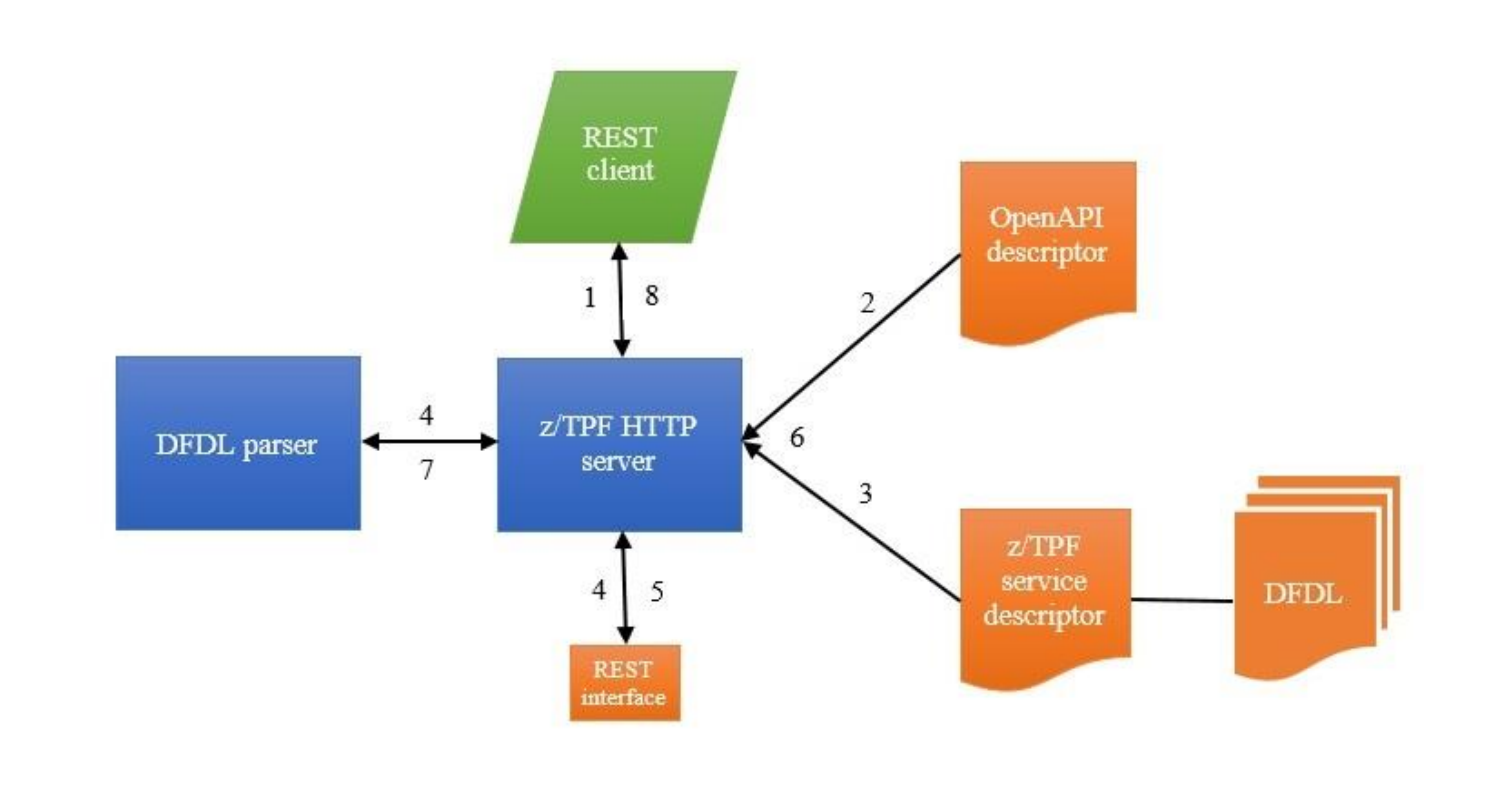
Create response handlers that:
1) build a single response structure
  - no HTTP, XML, JSON knowledge (JSON has preference)

int tpf_srvcSendResponse(tpf_srvc_token *token*,
    tpf_srvc_resp **response*, int *options*);

As-Is



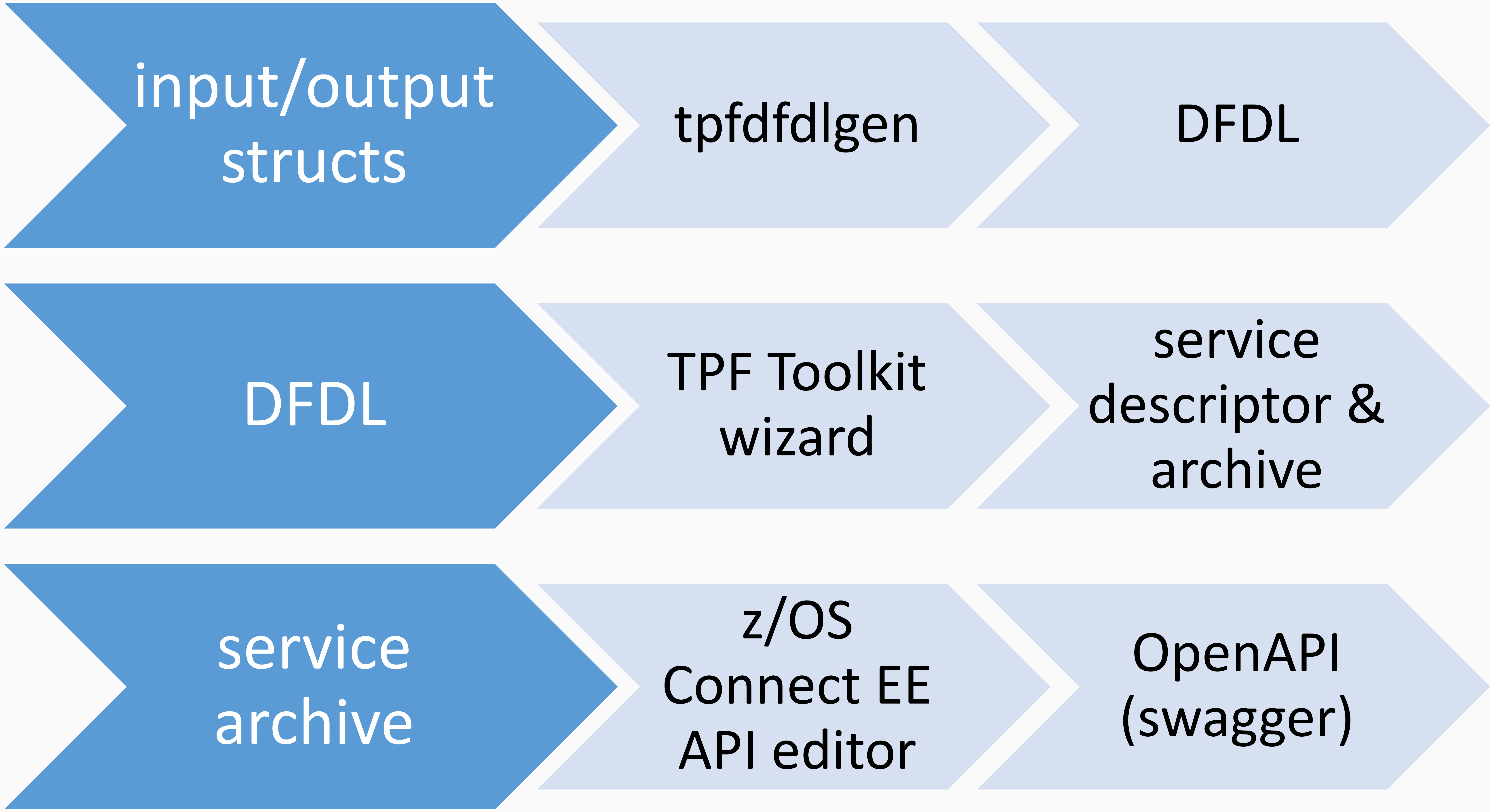
To-Be


REST client
OpenAPI descriptor
1
8
2
DFDL parser
4
7
z/TPF HTTP server
6
3
z/TPF service descriptor
DFDL
4
5
REST interface

REST Provider Support

- Tooling from TPF Toolkit and z/OS Connect provides simplified way of creating REST services without much code
- OpenAPI (aka Swagger) is used for REST implementation
- Demo in “TPF Toolkit” YouTube channel
- More details provided in Ongoing TPF Education

API Creation



input/output structs → tpfdfdlgen → DFDL

DFDL → TPF Toolkit wizard → service descriptor & archive

service archive → z/OS Connect EE API editor → OpenAPI (swagger)

Developer Process

1. Code service wrapper program: PRG1(<struct> *input, uint length, tpf_srvc_token tok);
2. Generate DFDL for input/output structures (maketpf PRG1 dfdl)
3. Describe the service (z/TPF service descriptor – TPF Toolkit)
4. Design REST API (OpenAPI descriptor – z/OS Connect EE API Editor)
5. Load to z/TPF (wrapper program, DFDL, service descriptor, OpenAPI descriptor)
6. Update URL program mapping file for HTTP server and deploy OpenAPI descriptor
7. Use swagger tooling to unit test REST API
8. Use swagger tooling to generate client code and/or documentation

1. Code service wrapper

```
void PRG1 (struct prg1_input *input, unsigned int in_len,
           tpf_srvc_token token) {

     struct prg1_output output;
     tpf_srvc_resp response;
     int rc = 0;

     // Do code to setup program call from input structure

     PRG2();          // call internal service

     // Do code to populate output structure

     response.status = TPF_SRVC_OK;
     response.data = &output;
     response.datalen = out_len;
     rc = tpf_srvcSendResponse(token, &response, 0);

    exit(0);
}
```

2. Generate DFDL

On z/linux:

```
> maketpf PRG1 dfdl
```

- Files are written to TPF_DFDL_DIR.
- File names are of format: <struct name>.gen.dfdl.xsd:

prg1_input.gen.dfdl.xsd

prg1_output.gen.dfdl.xsd

3. Describe service

Operation ID

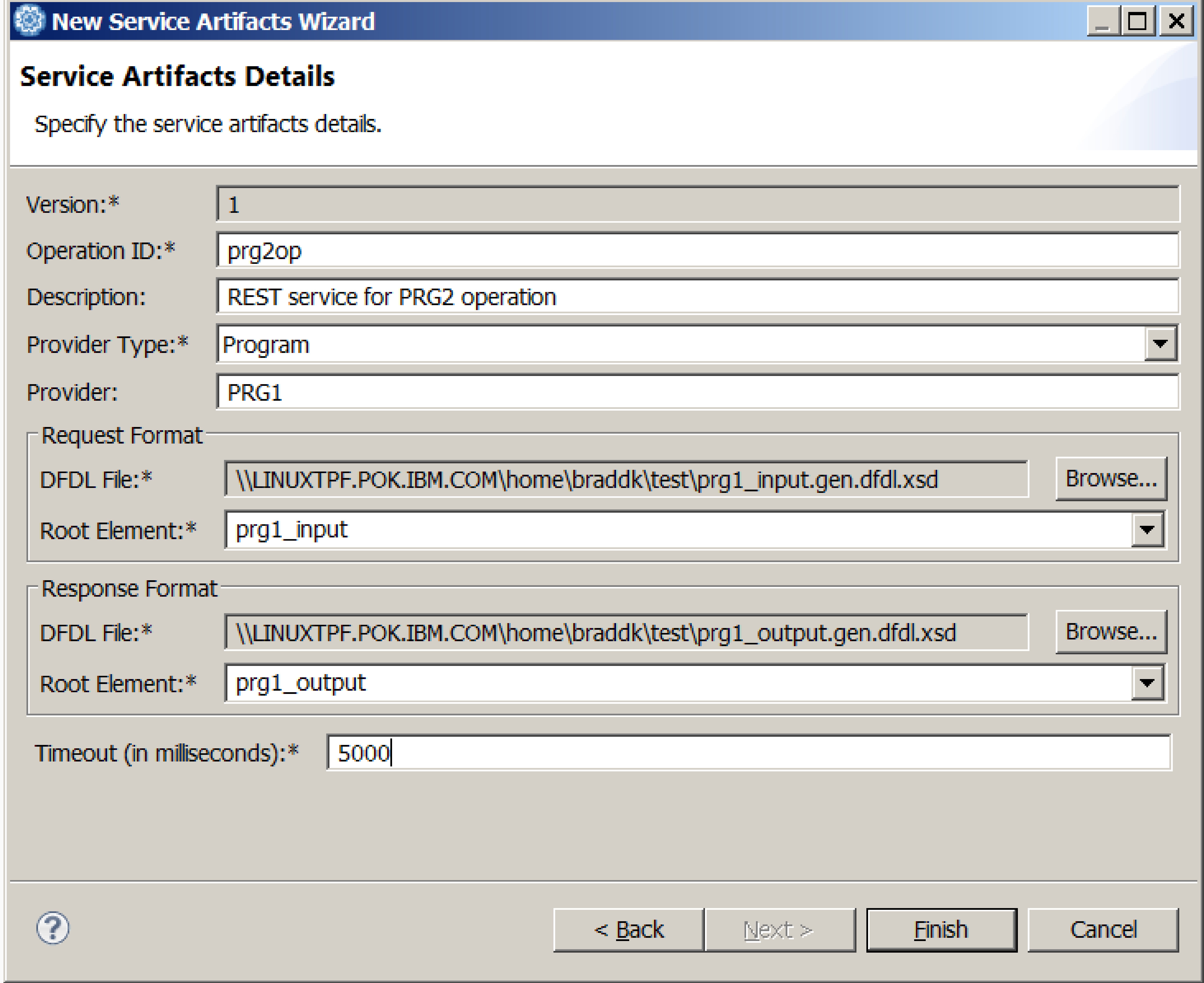
Provider Type

Provider

DFDL input structure

DFDL output structure

Timeout



New Service Artifacts Wizard

Service Artifacts Details

Specify the service artifacts details.

Version:* 1

Operation ID:* prg2op

Description: REST service for PRG2 operation

Provider Type:* Program

Provider: PRG1

Request Format

DFDL File:* \\LINUXTPF.POK.IBM.COM\home\braddk\test\prg1_input.gen.dfdl.xsd

Root Element:* prg1_input

Response Format

DFDL File:* \\LINUXTPF.POK.IBM.COM\home\braddk\test\prg1_output.gen.dfdl.xsd

Root Element:* prg1_output

Timeout (in milliseconds):* 5000

4. Design REST API

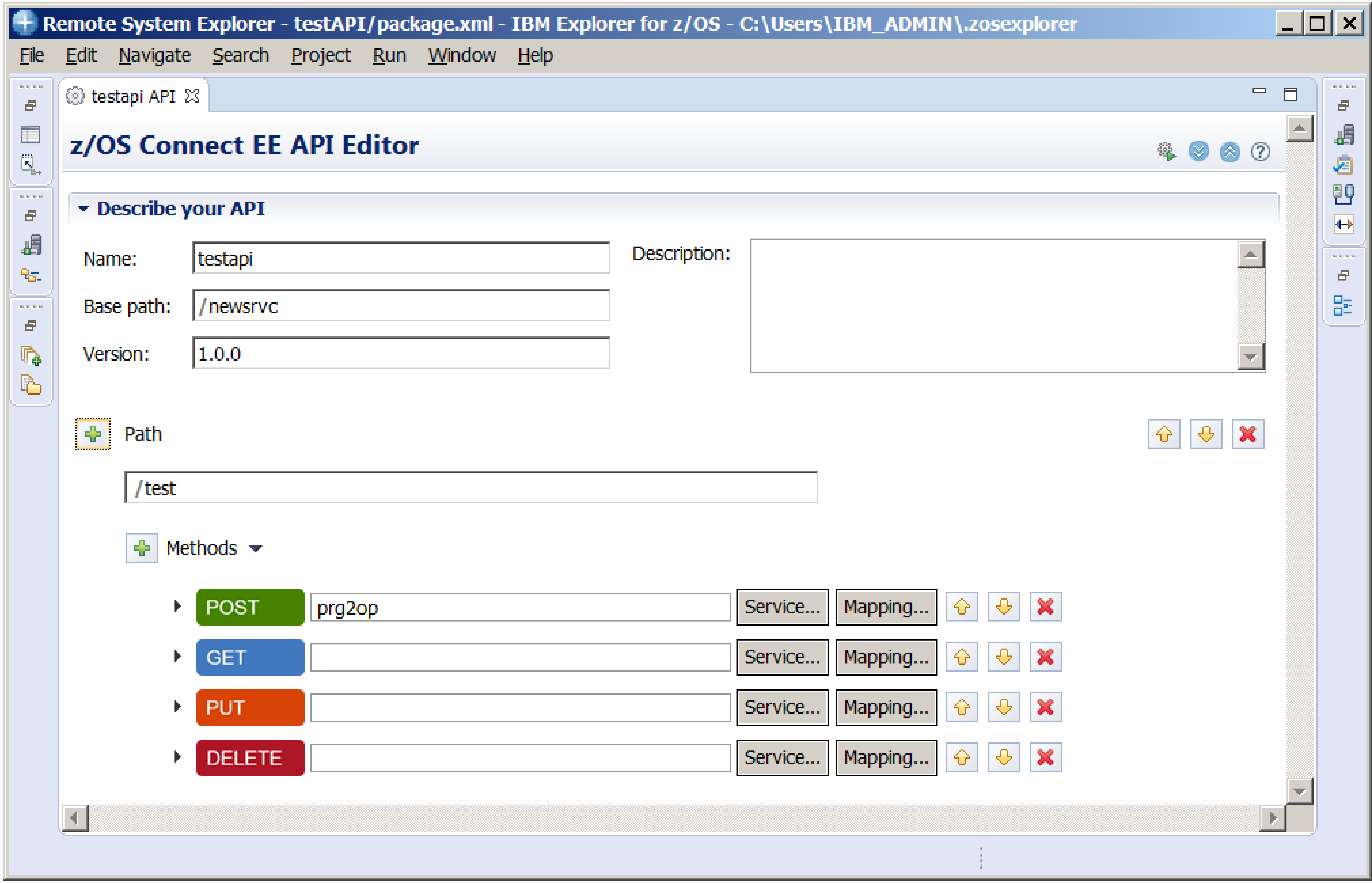
Name

Base path + Path = URI

Method

Operation ID

Free Download!



Remote System Explorer - testAPI/package.xml - IBM Explorer for z/OS - C:\Users\IBM_ADMIN\.zosexplorer
File Edit Navigate Search Project Run Window Help
testapi API
z/OS Connect EE API Editor
Describe your API

Name: testapi

Base path: /newsrvc

Version: 1.0.0

Description:

Path

/test

Methods

POST	prg2op	Service...	Mapping...
GET		Service...	Mapping...
PUT		Service...	Mapping...
DELETE		Service...	Mapping...

5. Load to z/TPF

1. Service wrapper program
   PRG1.so
2. DFDL
   /sys/tpf_pbfiles/tpf-fdes/prg1_input.gen.dfdl.xsd
   /sys/tpf_pbfiles/tpf-fdes/prg1_output.gen.dfdl.xsd
3. Service descriptor
   /sys/tpf_pbfiles/tpf-fdes/prg2op.srvc.json
4. REST API documentation
   /sys/tpf_pbfiles/tpf-fdes/newsrvc.swagger.json

6. Update URL mapping file

Update /etc/tpf_httpserver/url_program_map.conf on z/TPF

URL program mapping file format:

<Base path>* <OpenAPI filename>

Example:

/newsrvc* newsrvc.swagger.json

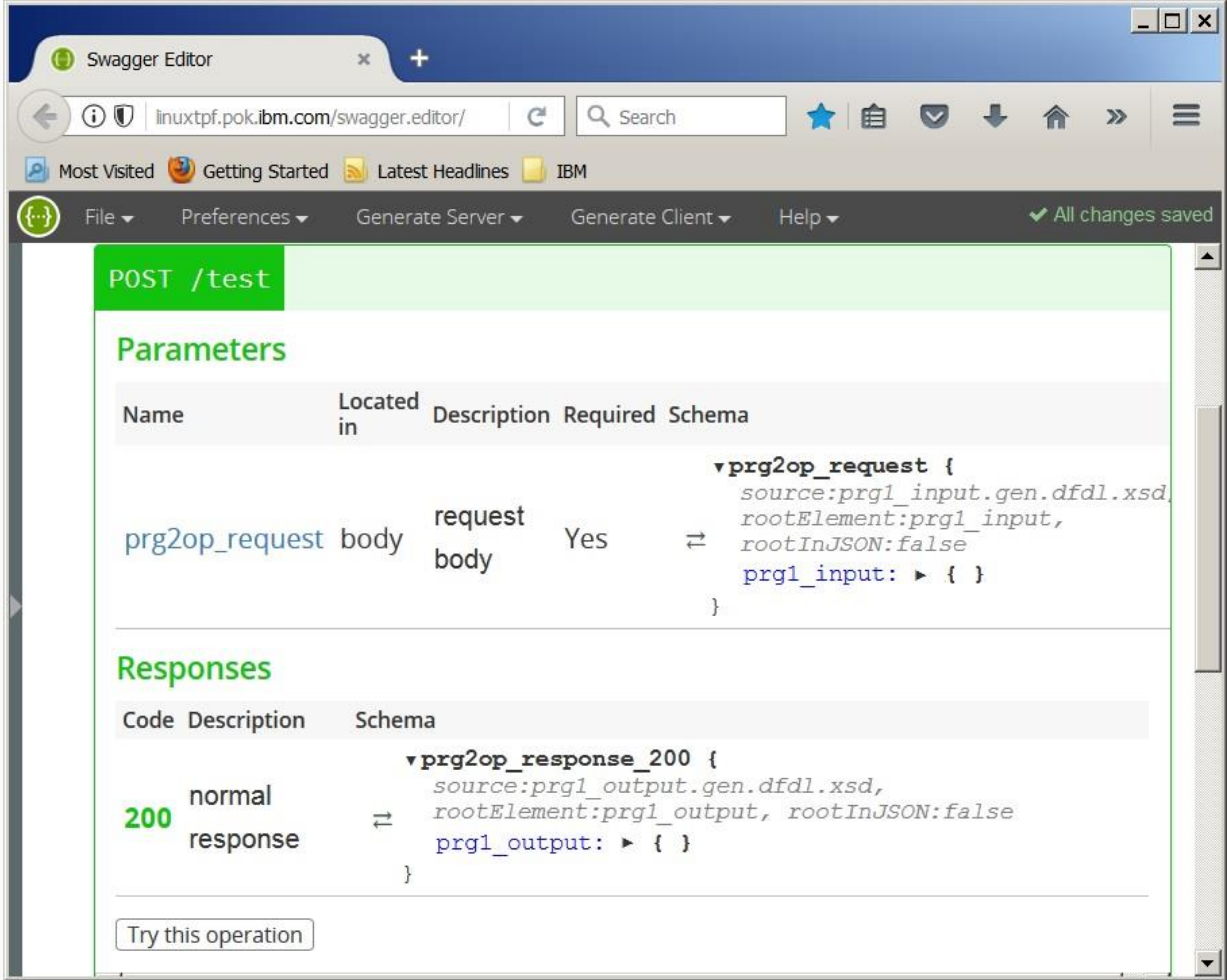
7. Use swagger tooling

Swagger Editor

Swagger UI

Swagger Codegen

http://swagger.io/tools



POST /test

Parameters

Name	Located in	Description	Required	Schema

prg2op_request	body	request body	Yes	⇄ ▾prg2op_request {
				source:prg1_input.gen.dfdl.xsd,
				rootElement:prg1_input,
				rootInJSON:false
				prg1_input: ► { }
				}

Responses

Code	Description	Schema

200	normal response	⇄ ▾prg2op_response_200 {
		source:prg1_output.gen.dfdl.xsd,
		rootElement:prg1_output, rootInJSON:false
		prg1_output: ► { }
		}

REST Provider Artifacts

- DFDL
- z/TPF service descriptor
- Service archive
- OpenAPI descriptor

DFDL (.dfdl.xsd)

- DFDL is required for the input and output structures to transform HTTP request/response.
- Includes data from HTTP query parameters, headers, and body.
- DFDL element names must match names used for parameters in HTTP interface.

DFDL conversion (request)

```
POST http://mytpf:81/newsrvc/test?parm1=90210
content-type: "application/json"
content-length: "250"
parm2: "76"
{ "prg1_input": .... }
```

```
struct prg1_input {         // same name or change in dfdl file to match "prg1_input"
   int parm1;               // same name or change in dfdl file to match
   short parm2;             //   must be a unique name for the dfdl file
   ...
};
```

DFDL conversion (response)

```
struct prg1_output {          // same name or change in dfdl file to match “prg1_output”
   short parm2;               // same name or change in dfdl file to match “parm2”
   ...
};
```

↓

```
Status Code: <status> <status_reason>
content-type: “application/json”
content-length: “123”
parm2: “76”
{ “prg1_output”: .... }
```

Service descriptor (.srvc.json)

- Requires subsystem unique operationId
  Used to associate REST API to service provider

- Determines where the service is located:
  Program – Assembler/C/C++
  JAM – application manager for Java

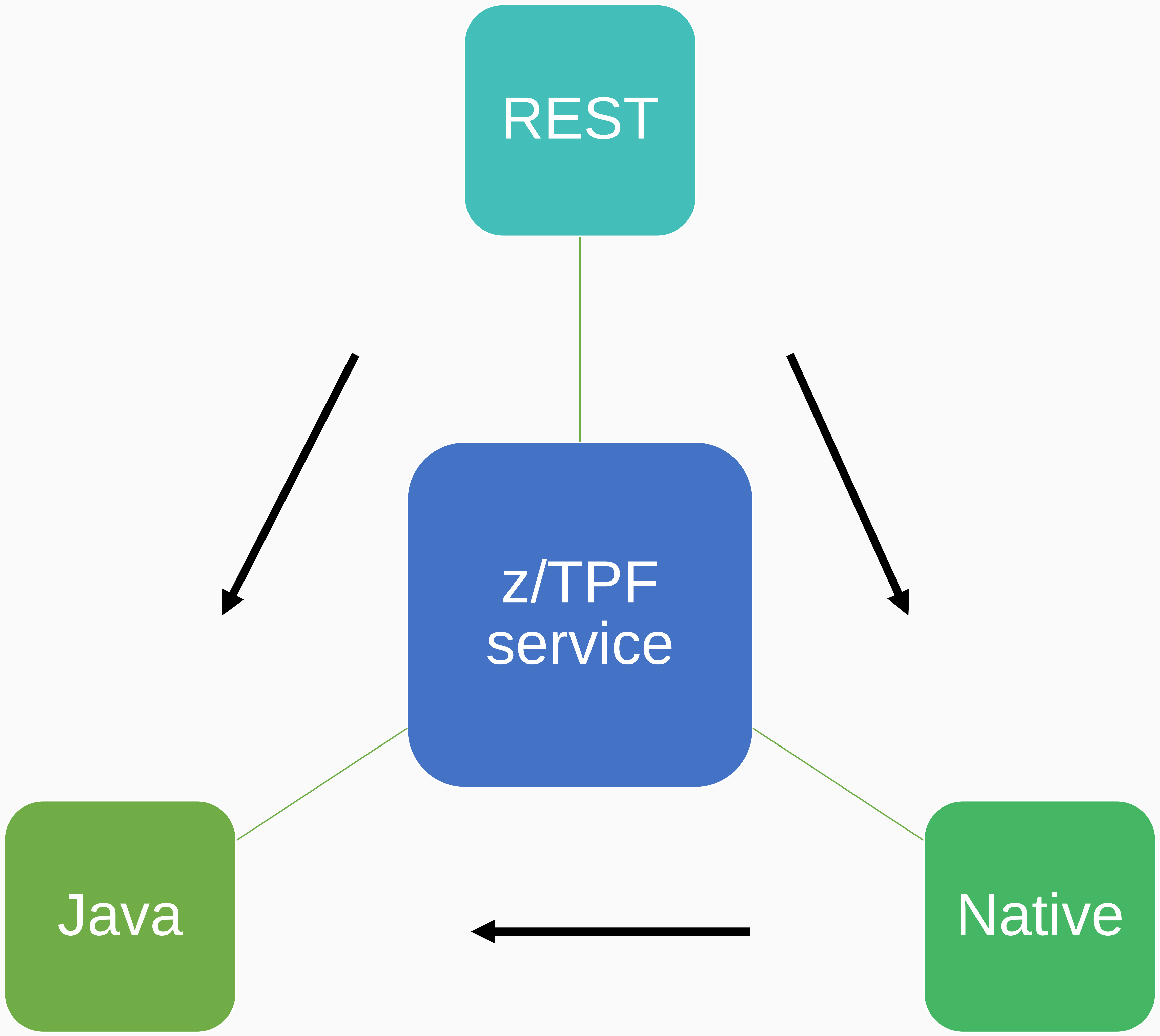
- Timeout – regardless of internal or external call

z/TPF service routing

REST call -> Native service

REST call -> Java service

Native call -> Java service



Service archive (.sar)

- Created by TPF Toolkit Service Artifacts Wizard

  - IBM internal format

- Required by z/OS Connect EE API Editor (creates OpenAPI descriptor)

  - Common tooling with CICS, IMS, z/OS, etc.

- Does not get loaded to z/TPF

OpenAPI (.swagger.json)

- Requires subsystem unique operationId per API
  Used to associate REST API to service provider (1-1 mapping)

- Describes the REST API details of the HTTP request/response
  host
  URI
  method
  parameters -> query, header, body
  response status codes

REST API request

```
POST http://mytpf:81/newsrvc/test?parm1=90210
content-type: “application/json”
content-length: “250”
parm2: “76”
{ “prg1_input”: .... }
```

↓

route to **operationId** (defined by z/TPF service descriptor)

REST API response

tpf_srvcSendResponse

- **tpf_srvc_token** -> associated with connection and operationId
- tpf_srvc_resp.data -> data for transformation
- tpf_srvc_resp.datalen
- tpf_srvc_resp.**status** -> HTTP status code
- tpf_srvc_resp.**status_reason** -> HTTP status text

↓

Status code: <**status**> <**status_reason**>
parm2: ...
{“prg1_output”: ... }

DFDL with OpenAPI

Q. What happens for elements defined in DFDL but not OpenAPI?

A. During transformation to data (BLOB), they receive the DFDL default value.
During transformation to HTTP, they are not included in the response (ignored).

Q. What happens for parameters defined in OpenAPI but not in DFDL?

A. On either a request/response they are ignored.
A warning message will occur during OpenAPI deployment.

Q. What happens when the element/parameter is defined in both but not in HTTP?

A. The OpenAPI default value is used (if provided) else the DFDL default value is used.

REST Provider APIs

- API Discovery
- z/OS Connect APIs
- z/TPF system APIs

API Discovery

GET <basePath>/api-docs

returns the OpenAPI document

- Useful for tools that can import OpenAPI docs by URL such as API Connect (management & security) and Swagger Editor/Swagger UI (unit test).
- The “host” field in the OpenAPI descriptor is updated with the host and port values used for the request.
- Requires:

    1) URL program mapping file updated with OpenAPI descriptor and basePath.

    2) OpenAPI descriptor deployed through common deployment

z/OS Connect APIs

GET /zosConnect/apis

returns list of OpenAPI descriptors loaded to system

GET /zosConnect/apis/<OpenAPI filename>

returns information about the OpenAPI descriptor

z/OS Connect admin APIs are currently not implemented

- start/stop services
- create/update/delete services

z/TPF system APIs

- Reserved z/TPF system URIs:

  /tpf...

  /zosConnect...

- z/TPF may be releasing REST APIs documented and implemented through OpenAPI.
- Enablement would require:
  1) updating the URL-program mapping file
  2) deploying the OpenAPI descriptor
  3) starting/running the z/TPF HTTP server

REST consumer support

- Provide configurable REST consumer support that doesn't require code for HTTP, JSON, or XML
- OpenAPI (aka Swagger) is used for REST implementation
- Integrate calling of Java services on z/TPF and REST services off z/TPF

HTTP client support

Create a REST call that:

1) builds a HTTP request structure
2) creates the XML or JSON
3) provides connection information
4) consumes a HTTP response structure
5) parses the XML or JSON in the response

REST consumer support

Create a REST call that:

1) builds a single request structure
   - no HTTP, XML, JSON knowledge (JSON has preference)
   - connection information is provided by configuration

2) consumes a single response structure
   - no HTTP, XML, JSON knowledge

REST consumer support

- Uses same API used to call Java (tpf_srvcInvoke)
  - Allows easy migration of a Java service on/off z/TPF
  - No application change needed!
- Implemented through OpenAPI & z/TPF service desciptor

To-Be

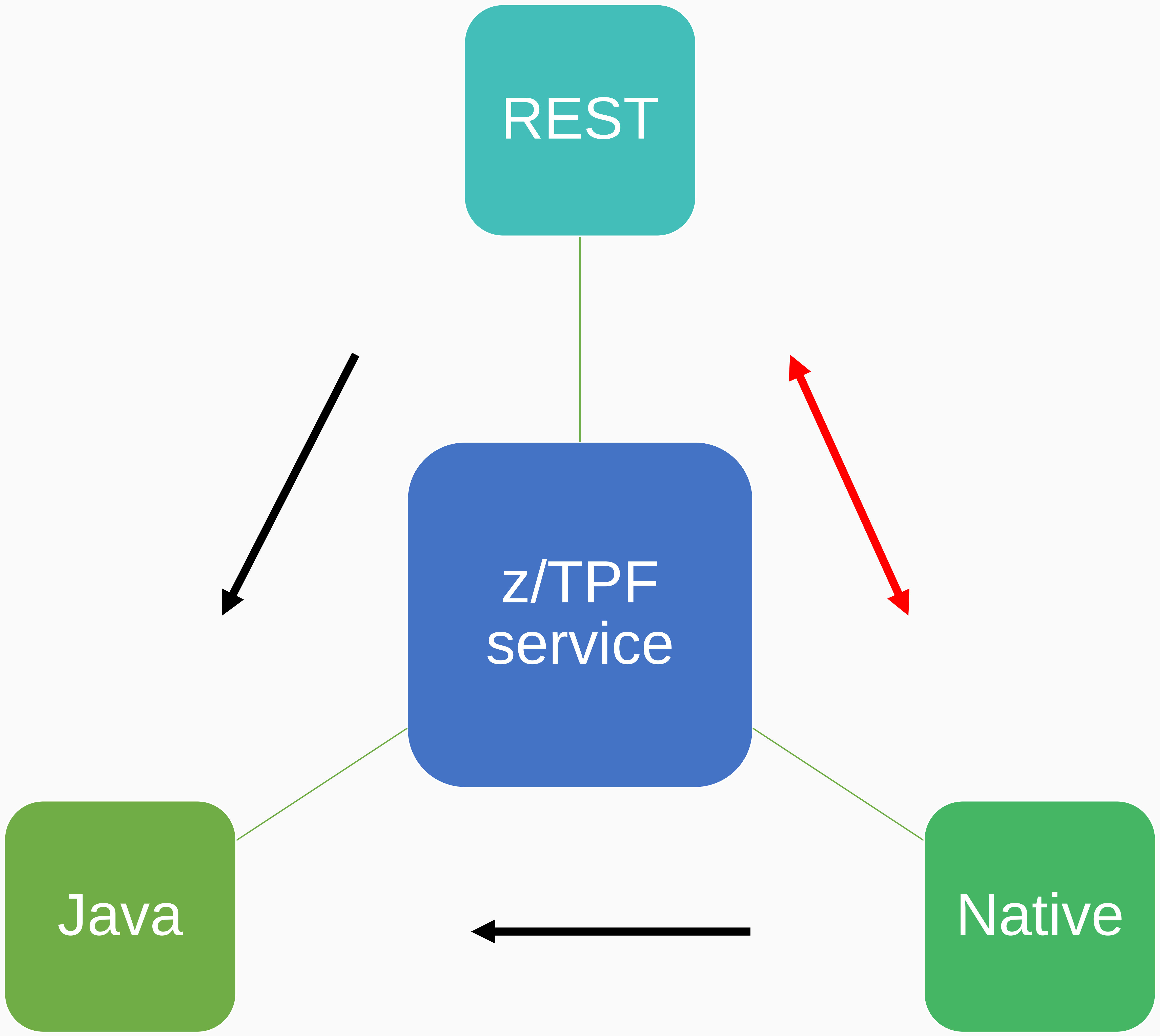
z/TPF service routing

REST call -> Native service

REST call -> Java service

Native call -> Java service

Native call -> REST service



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.