



REST Provider Education

Education

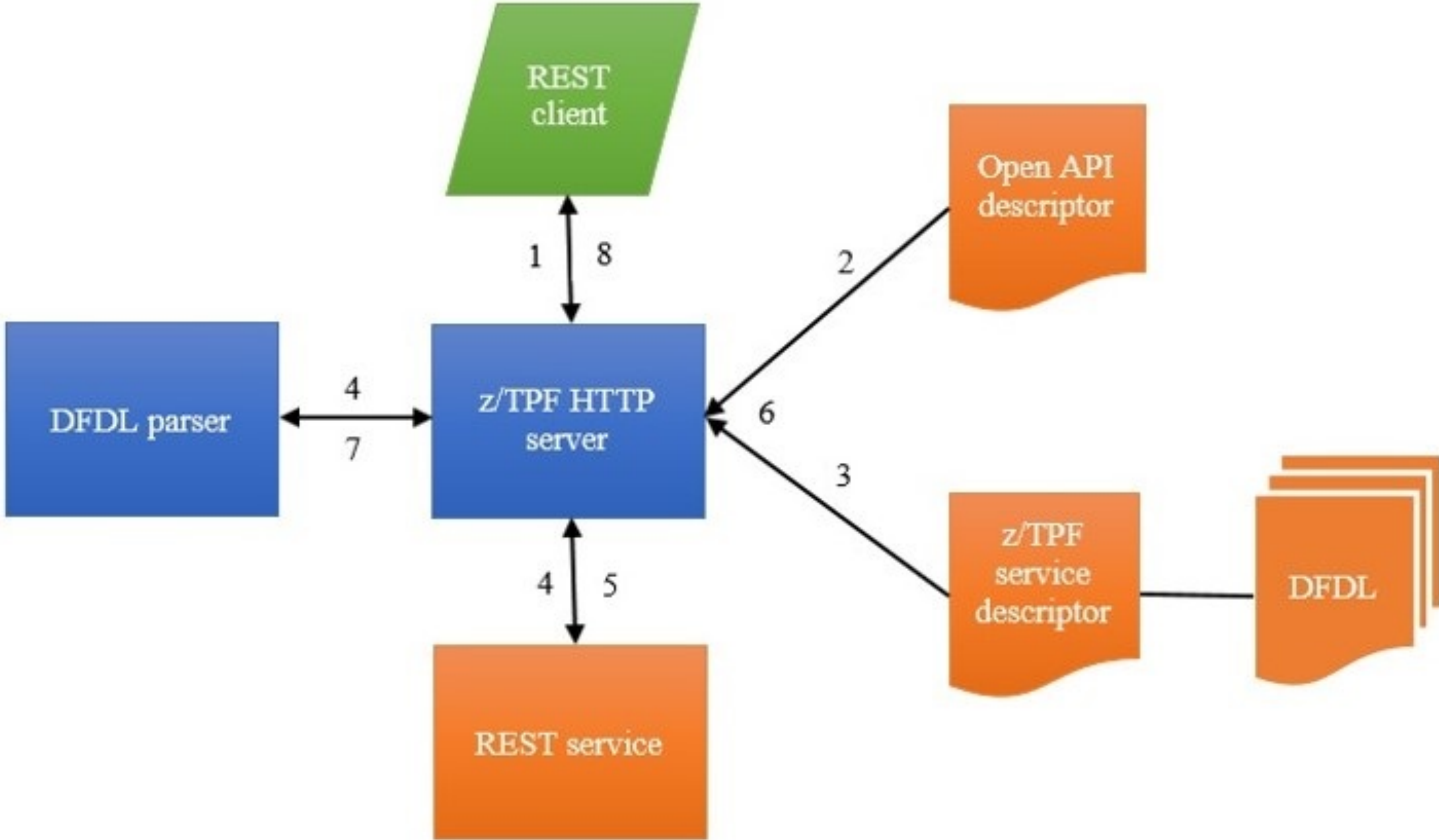
Bradd Kadlecik
z/TPF Development

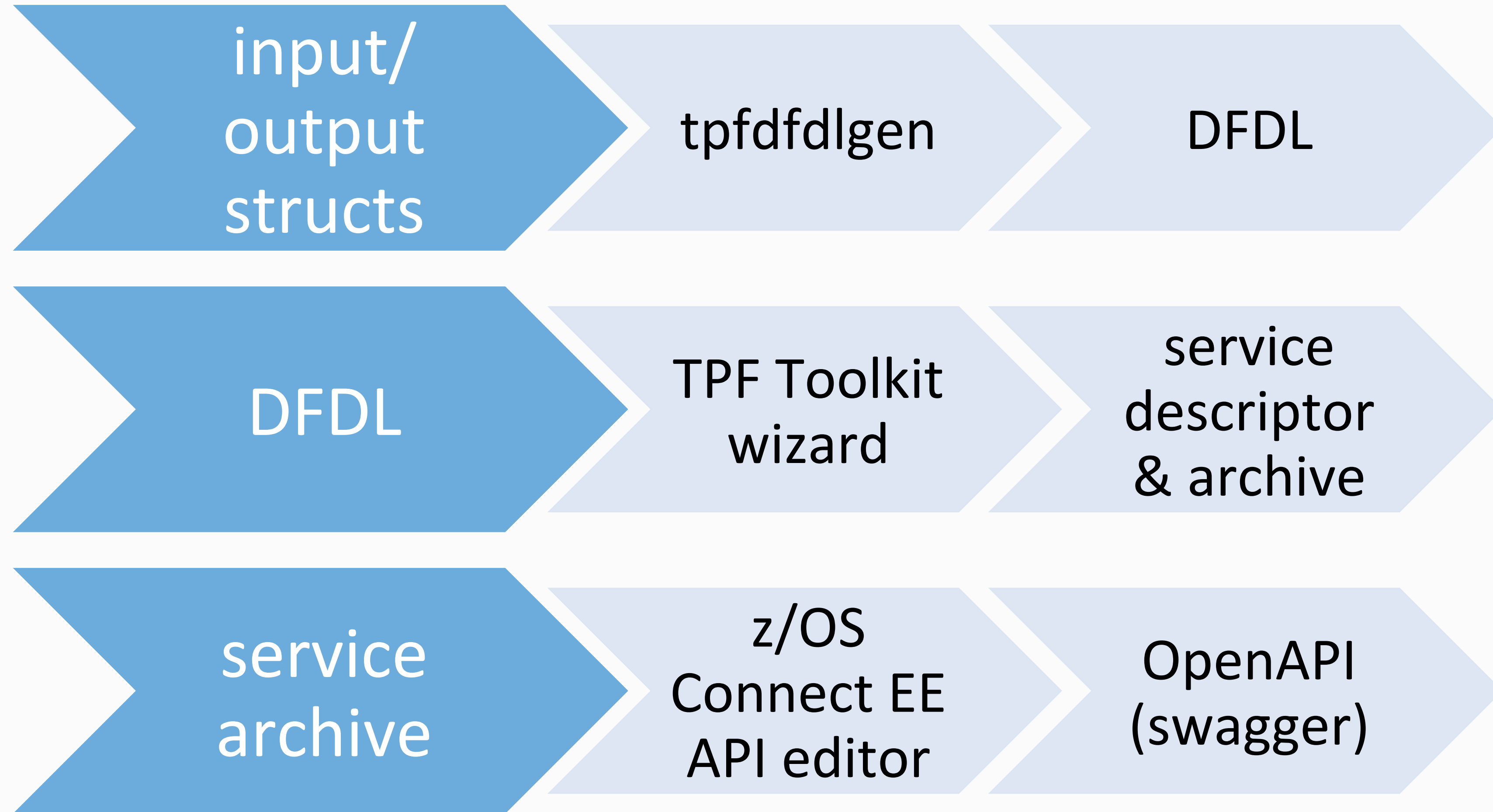
© 2017 IBM z/TPF | TPF Users Group Spring Conference | IBM Confidential

IBM z/TPF
April 5th, 2017

- Overview
- API Creation
- API Testing & Management

Overview





API Creation

- Tooling from TPF Toolkit and z/OS Connect provides simplified way of creating REST services without much code
- Demo in “TPF Toolkit” YouTube channel

Before you begin

TPF Toolkit 4.2.9 installed

IBM Explorer for z/OS installed

Free download from IBM developerWorks

<https://developer.ibm.com/mainframe/products/downloads/eclipse-tools/>

YouTube IBM TPF Toolkit channel

<https://www.youtube.com/watch?v=p1eD9b5VpA8>

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);  
}
```


API Creation

```
#include <string.h>
#include <tpf/c_eb0eb.h>
#include <tpf/c_https.h>
#include <restDFEDHeader.h>
#include <tpf/services.h>

//Wrapper Function for POST Request

extern "C" void addPassenger(qxhf_parms* newPassenger, unsigned int reqsize,
                             tpf_srvc_token http_token) {

    tpf_srvc_resp response;
    char buffer [256]; //USE this to send Status Messages

    ecbptr()->ebxsw0 = 0x01; //Set mode to func return

    newPassenger->autodix = 1; //Use autoindexing

    //Use the dfed function corresponding to ZTEST DFED FLIGHT ADD PAS ...

    qxhf_addpsn(newPassenger);

    //Check Error Status

    unsigned int ret;

    memcpy(&ret, &ecbptr()->ebrs01, sizeof(ret));
```

API Creation

```
if (ret) {
    //Sends a negative response
    response.data=NULL;
    response.datalen=0;
    response.status=IHTTPS_STATUS_400;

    sprintf (buffer, "The DFED parameter for FLIGHT was not ADDED check ERROR
NUMBER: %d in qxhf_addpsn.cpp to debug", ret);

    response.status_reason=buffer;

    tpf_srvcSendResponse (http_token, &response, 0);
}
else {
    newPassenger->detail = 1;          //Get details
    qxhf_dsppsn (newPassenger);
    memcpy (&ret, &ecbptr ()->ebrs01, sizeof (ret));

    if (ret) {

        response.data=NULL;
        response.datalen=0;
        response.status=IHTTPS_STATUS_401;
    }
}
```

API Creation

```
    sprintf (buffer, "The Flight was added sucessfully but can not return
a proper message back. Check ERROR NUMBER: %d in qxhf_dspps.c to debug", ret);

    response.status_reason=buffer;

    tpf_srvcSendResponse (http_token, &response, 0);
}

else{

    qxhf_parms *flightRec = NULL;

    memcpy (&flightRec, &ecbptr()->ebx008, sizeof(flightRec));

    //Send Success Response
    response.data=flightRec;
    response.datalen=sizeof(*flightRec)+1;
    response.status=IHTTPS_STATUS_200;
    response.status_reason=IHTTPS_STATUS_200_TEXT;

    tpf_srvcSendResponse (http_token, &response, 0);

    //Unallocate the memory we used from previous call.

    free(flightRec);

}

}

}
```

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

maketpf generation

```
braddk@linuxtpf:~/flight> maketpf qff1 dfdl
```

```
MTPF2210I: Configuration file: "/home/braddk/flight/./maketpf.cfg"
```

```
MTPF2111I: Makefile found: "/home/braddk/flight/dfed/QFF1.mak"
```

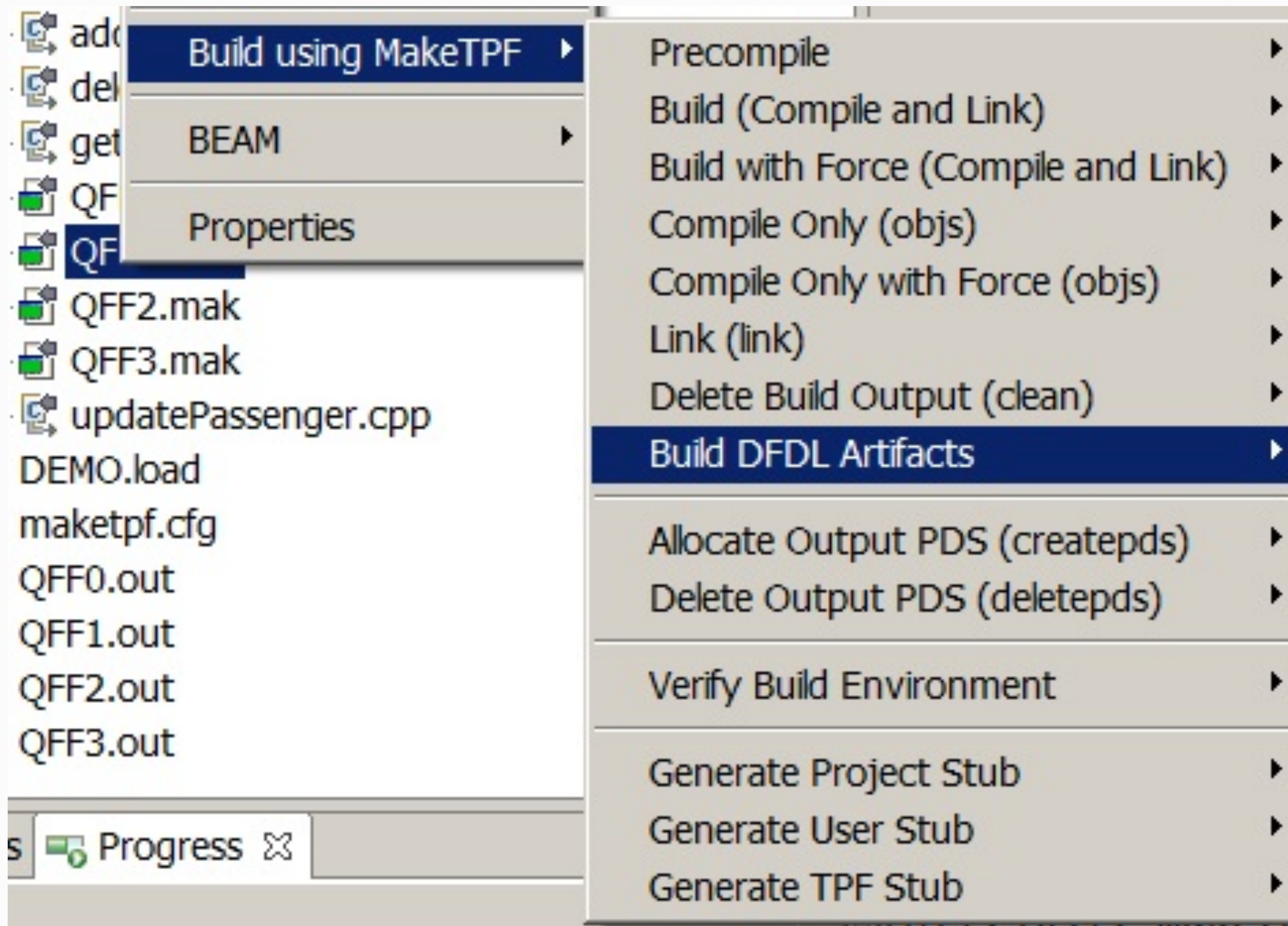
```
MTPF2109I: maketpf: "QFF1"
```

```
Command: "gnumake -f /home/braddk/flight/dfed/QFF1.mak -I /ztpf/  
commit/tpftools/include_ztpf_user -I /ztpf/commit/tpftools/include_ztpf -I /ztpf/  
cur/tpftools/include_ztpf_user -I /ztpf/cur/tpftools/include_ztpf -I /ztpf/tools/  
include_ztpf_user -I /ztpf/tools/include_ztpf dfdl 2>/home/braddk/flight/QFF1.err  
| tee /home/braddk/flight/QFF1.out"
```

```
mkdir -p /home/braddk/flight/bss/dfdlgen
```

```
tpfdfdlgen /home/braddk/flight/dfed/obj/addPassenger.o -d /home/braddk/flight/bss/  
dfdlgen
```

```
DFDLGEN0015I: DFDL files generated in /home/braddk/flight/bss/dfdlgen/
```



Verify the DFDL

- Are the generated DFDL names the ones you want to use for REST?
- Are the generated DFDL types correct? Is a char a string or numeric?
- Do you want to allow string padding? Is the padding NULL (%NUL;) or space(%SP;)?
- Are any strings required to be NULL terminated (dfdl:trailingSkip="1")?
- In case of a unions, is there a discriminator expression that can be used to determine the layout?
- Is there a variable length string that needs a variable length expression?

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 Browse...

Root Element:* prg1_input

Response Format

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

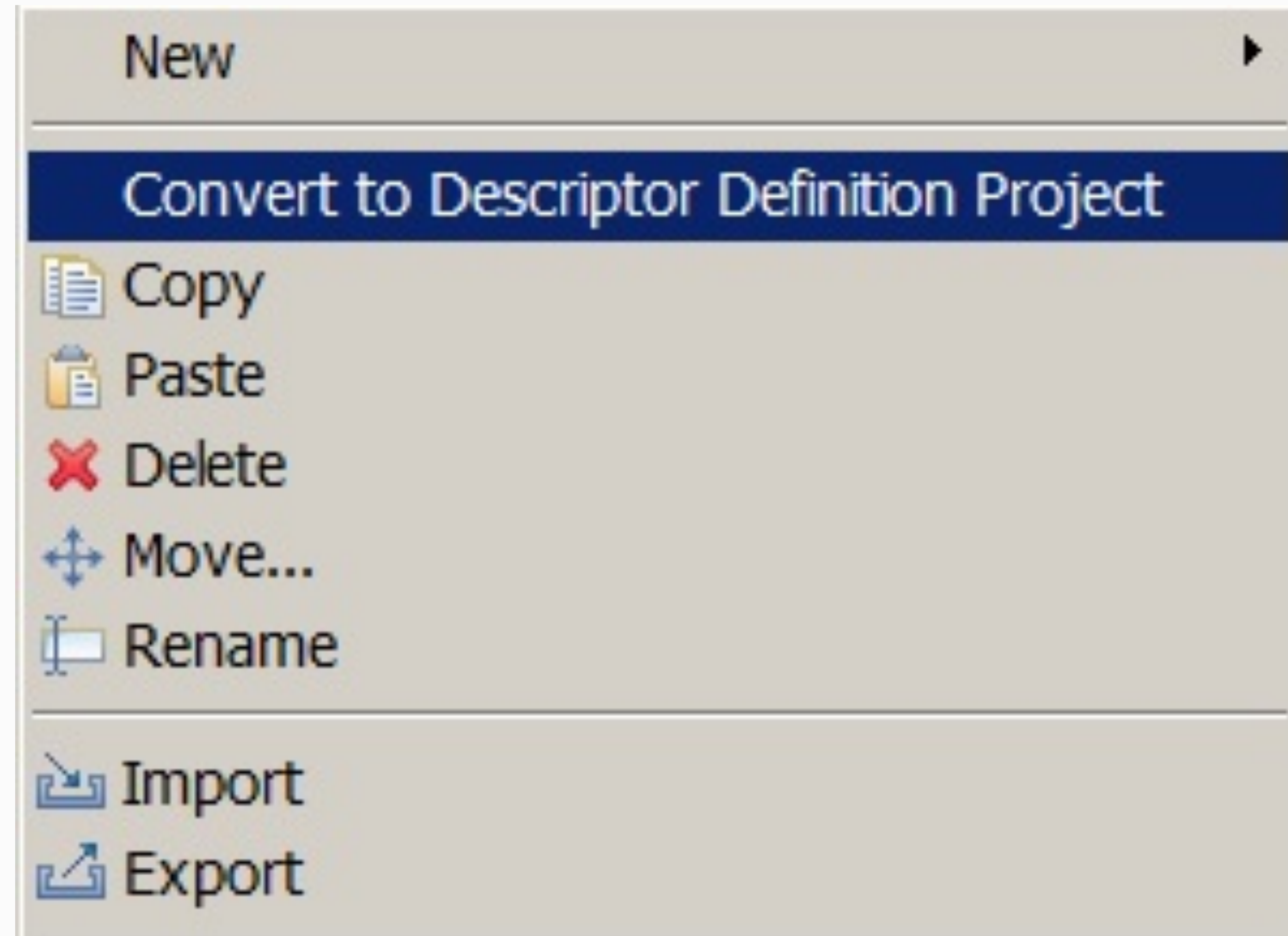
Root Element:* prg1_output

Timeout (in milliseconds):* 5000

? < Back Next > Finish Cancel

TPF Toolkit

Convert a TPF Project to a descriptor definition project.



TPF Toolkit

Import the request and response DFDL files into the Descriptor Definition Project...

Open Load File

Load using LoadTPF

OLDR load and activate loadset

OLDR deactivate and delete loadset

Import...

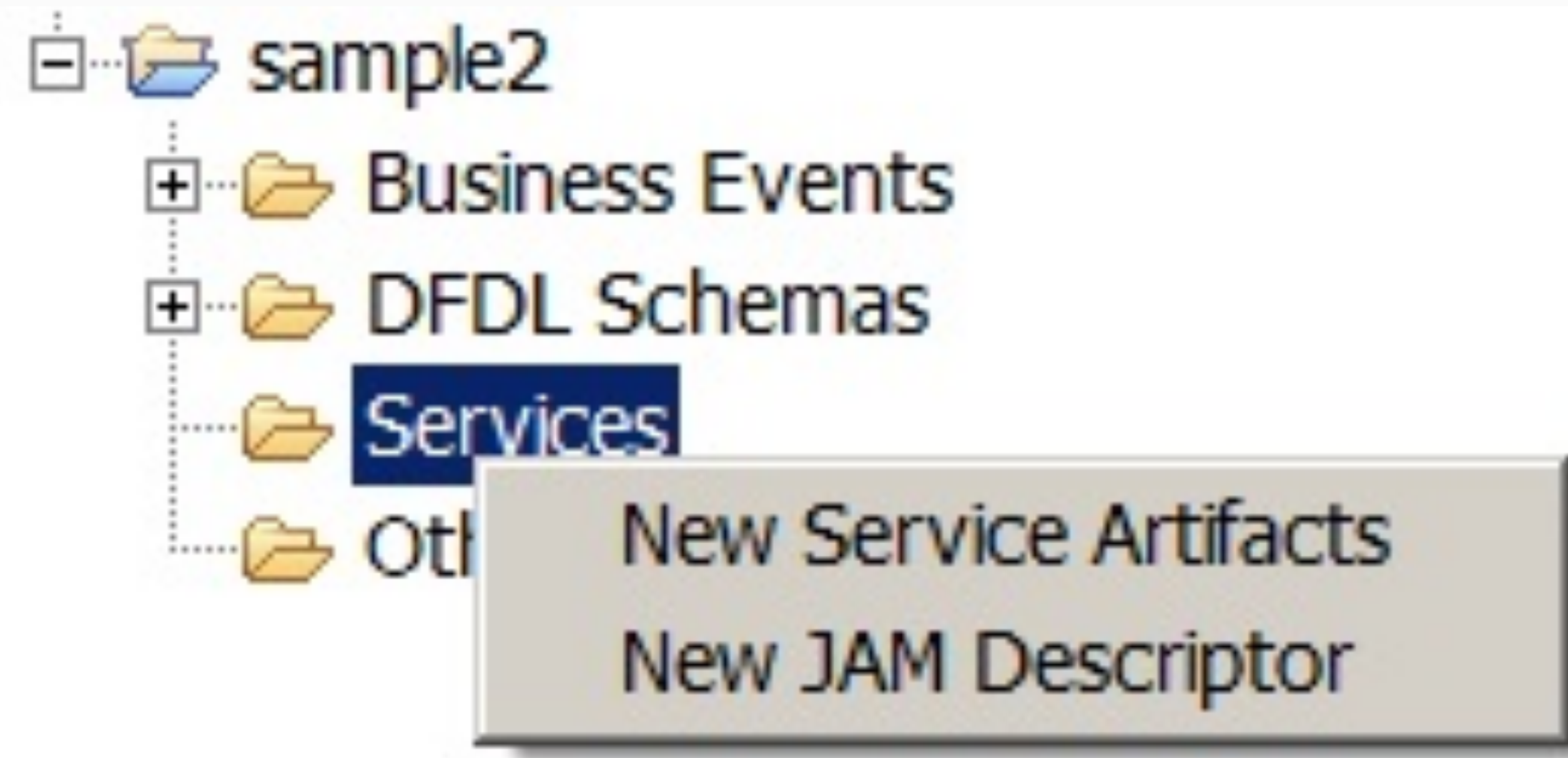
Refresh

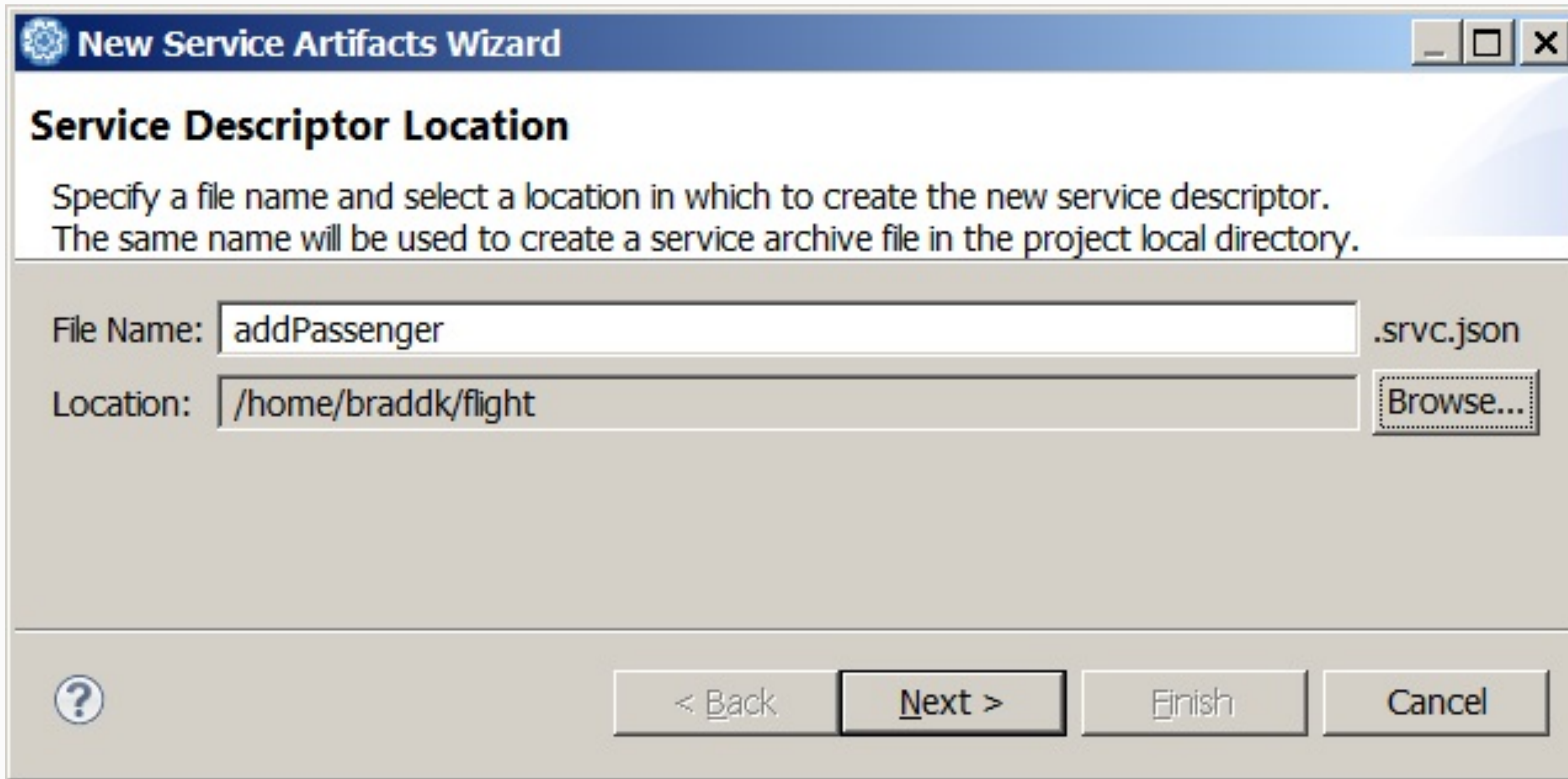
TPF Toolkit

Descriptor Definition project contains multiple wizards.

Create a service descriptor...

Right click on Services...





New Service Artifacts Wizard

Service Descriptor Location

Specify a file name and select a location in which to create the new service descriptor.
The same name will be used to create a service archive file in the project local directory.

File Name: .srvc.json

Location:

New Service Artifacts Wizard

Service Artifacts Details

Specify the service artifacts details.

Version:*

Operation ID:*

Description:

Provider Type:*

Provider:

Request Format

DFDL File:*

Root Element:*

Response Format

DFDL File:*

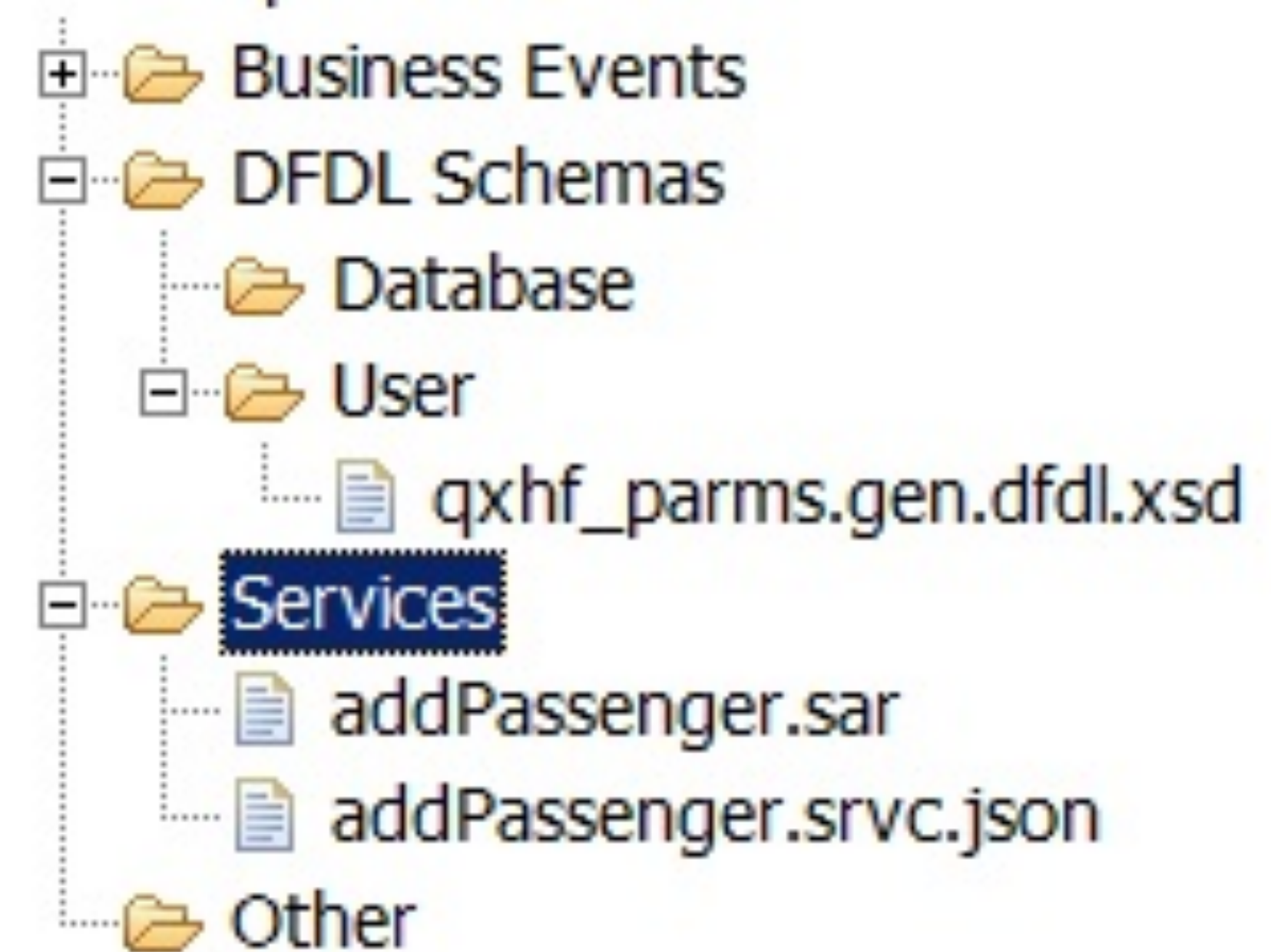
Root Element:*

Timeout (in milliseconds):*

TPF Toolkit

After clicking Finish, the service descriptor JSON file and service archive file is created.

The `.ssvc.json` file is in your project directory while the `.sar` file is in your local workspace.



addPassenger.srvc.json

```
{
  "version": 1,
  "operationId": "addPassenger",
  "description": "Service Descriptor definition for adding a passenger record",
  "providerType": "Program",
  "provider": "QFF1",
  "request": {
    "schema": "qxhf_parms.gen.dfdl.xsd",
    "root": "qxhf_parms"
  },
  "response": {
    "schema": "qxhf_parms.gen.dfdl.xsd",
    "root": "qxhf_parms"
  },
  "timeout": 10000
}
```

4. Design REST API

Name

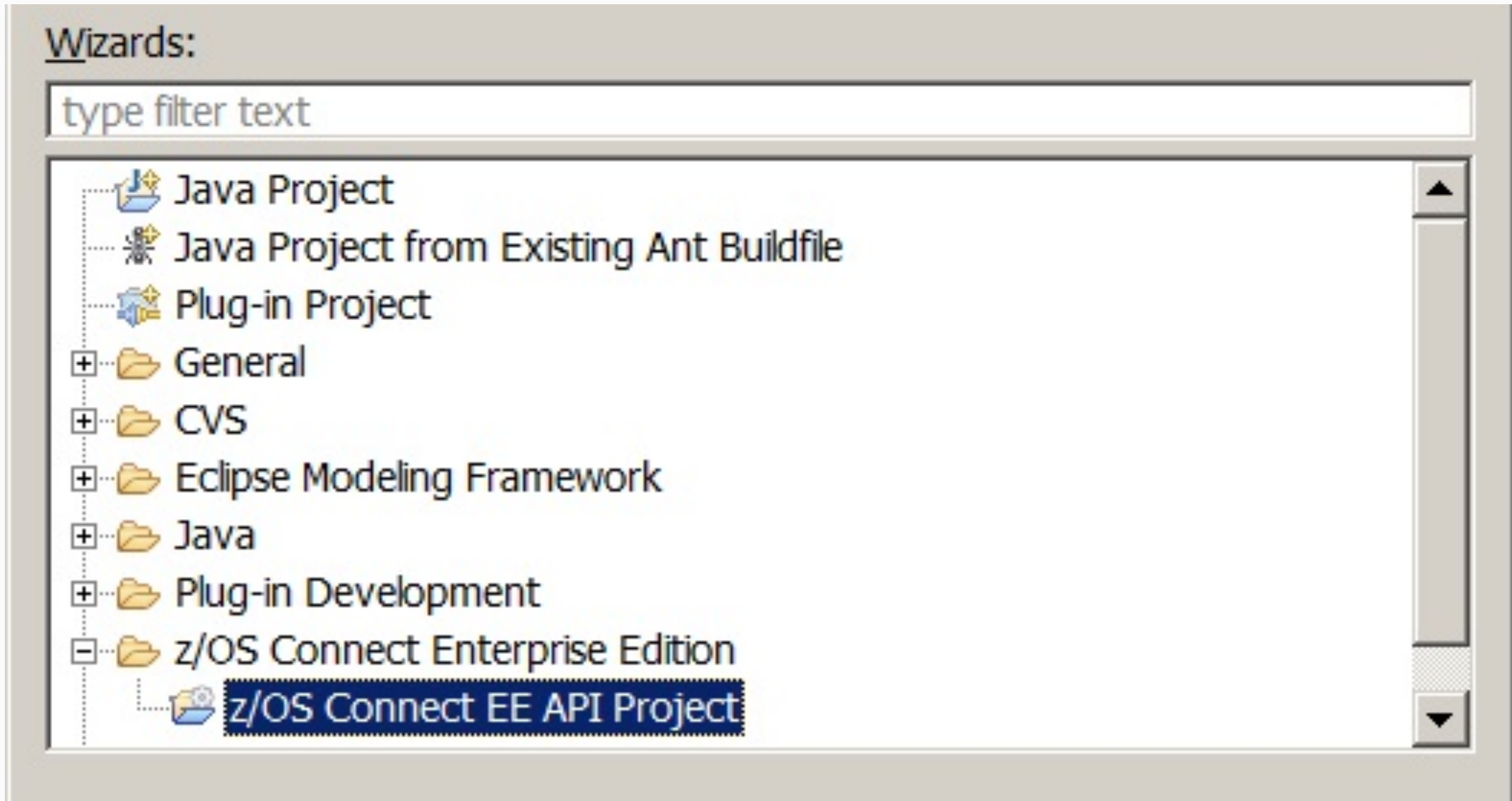
Base path + Path = URI

Method

Operation ID

The screenshot shows the 'z/OS Connect EE API Editor' window. The title bar indicates the file path: 'testAPI/package.xml - IBM Explorer for z/OS - C:\Users\IBM_ADMIN\.zosexplorer'. The menu bar includes 'File', 'Edit', 'Navigate', 'Search', 'Project', 'Run', 'Window', and 'Help'. The main workspace is titled 'testapi API' and contains the following configuration:

- Describe your API:**
 - Name: testapi
 - Base path: /newsrvc
 - Version: 1.0.0
 - Description: (empty text area)
- Path:** /test
- Methods:**
 - POST prg2op (Operation ID) with Service... and Mapping... buttons.
 - GET (empty Operation ID) with Service... and Mapping... buttons.
 - PUT (empty Operation ID) with Service... and Mapping... buttons.
 - DELETE (empty Operation ID) with Service... and Mapping... buttons.



Project name:

dfed

API name:

DFED Transaction Processing API

Base path:

/dfed

Description:

TPF flight, hotel, credit transaction processing system exposed through this REST interface.

API Editor

Click on Service...
Select the .sar file
Can use properties
tab in TPF Toolkit
to locate where the
.sar was stored.

The screenshot shows the 'z/OS Connect EE API Editor' window. The title bar indicates the current project is 'dfed/package.xml' in 'IBM Explorer for z/OS'. The main area is titled 'Describe your API' and contains the following fields:

- Name:** DFED Transaction Processing API
- Description:** TPF flight, hotel, credit transaction processing system exposed through this REST interface.
- Base path:** /dfed
- Version:** 1.0.0

Below these fields, there is a 'Path' section with a text input field containing '/flight/passenger'. To the right of this field are three icons: an up arrow, a down arrow, and a red 'X'.

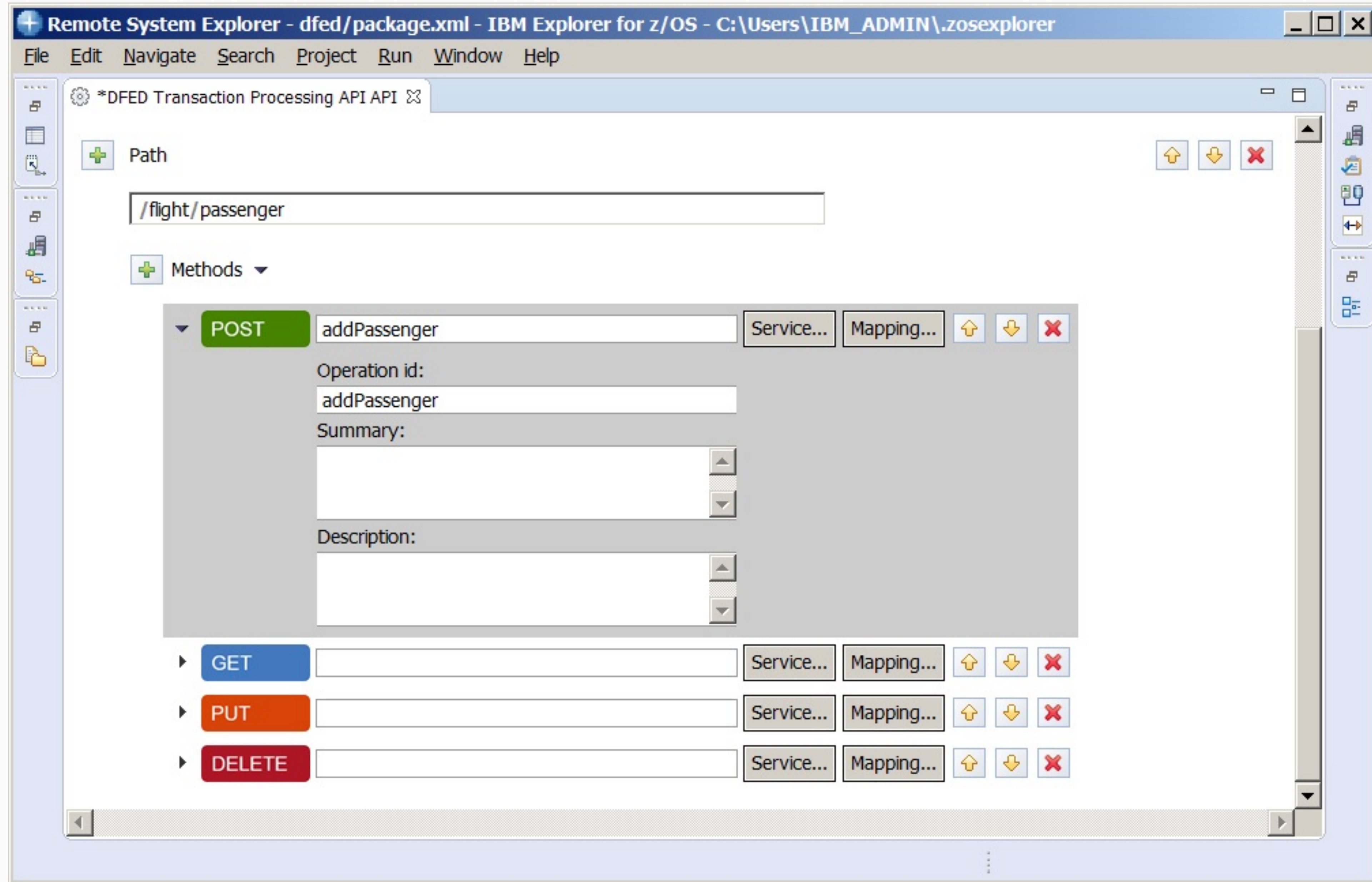
The 'Methods' section is expanded, showing a list of HTTP methods with their corresponding actions:

Method	Action	Service...	Mapping...	Up	Down	Delete
POST	addPassenger	Service...	Mapping...	↑	↓	✖
GET		Service...	Mapping...	↑	↓	✖
PUT		Service...	Mapping...	↑	↓	✖
DELETE		Service...	Mapping...	↑	↓	✖

API Editor

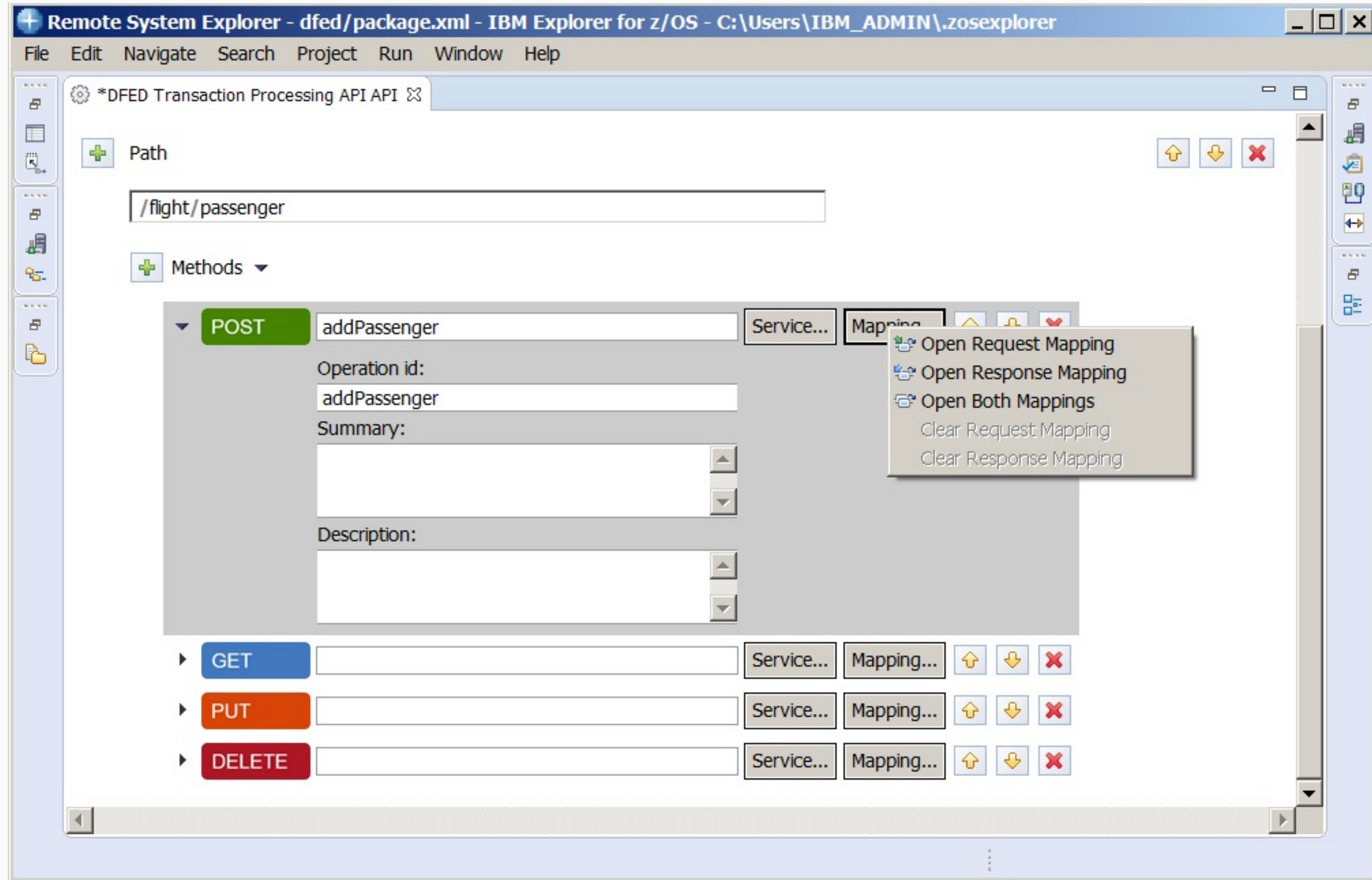
Expand method to update operationId to match service descriptor.

For this example postAddPassenger was changed to addPassenger.



API Editor

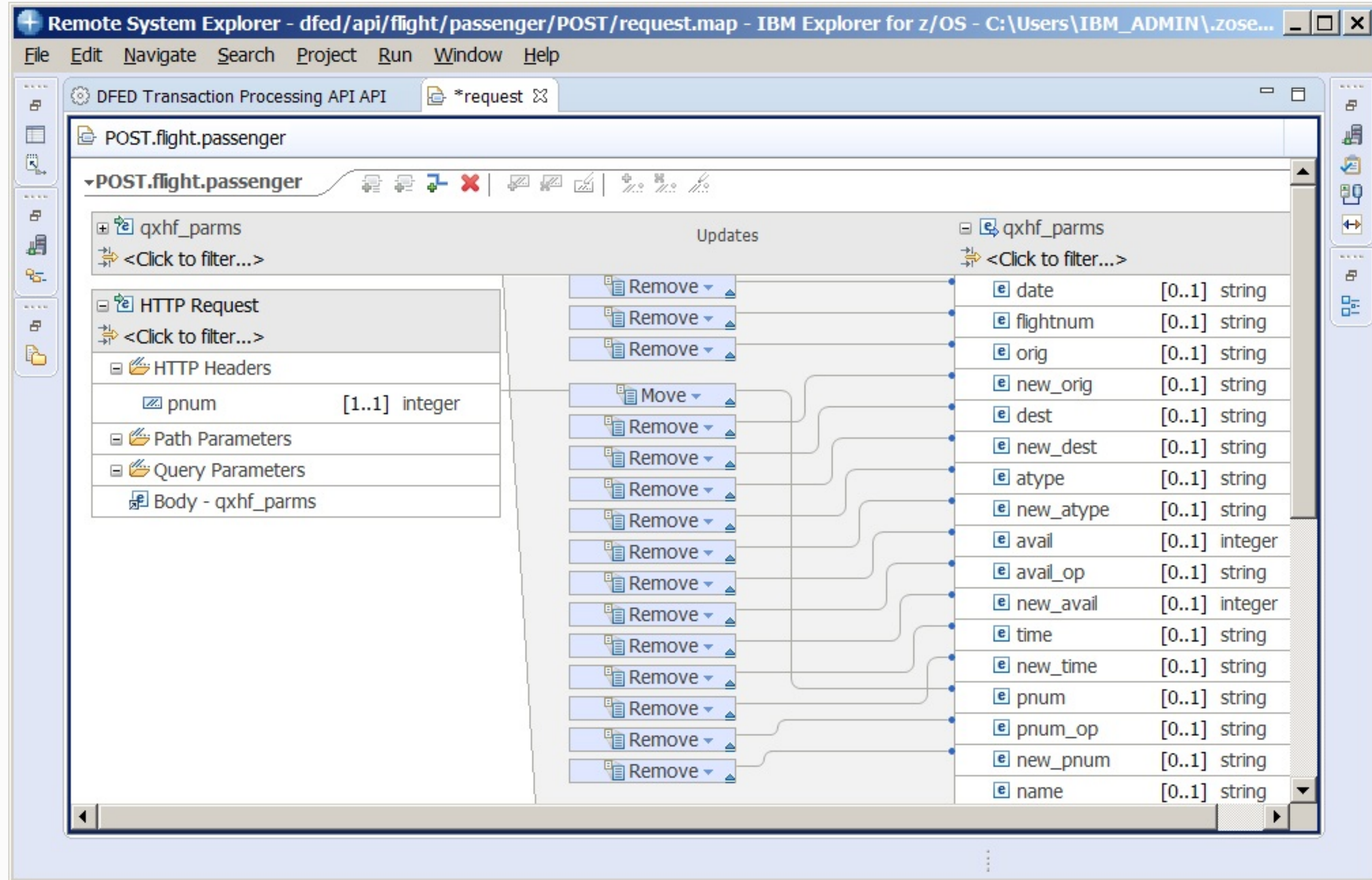
Use Mapping for request and response formats to specify query or header parameters or eliminate fields.



API Editor

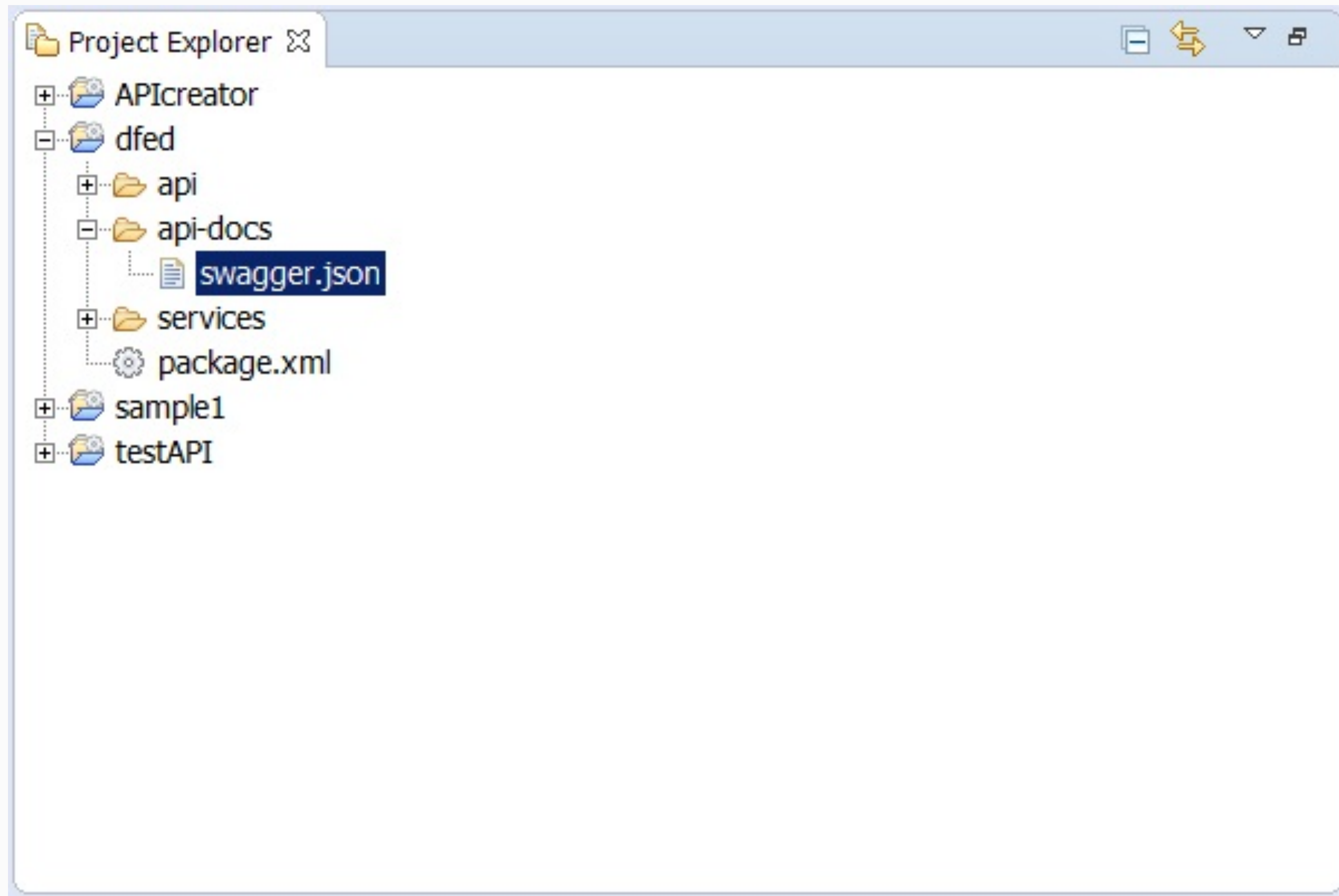
In the request mapping, delete the Authorization header.

Use Add Connection or Add Remove transform. Don't use Add assign transform or path parameters.



API Editor

Use Project Explorer to export swagger.json to TPF Project directory.



5. Load to z/TPF

1. Service wrapper program
QFF1.so
2. DFDL
/sys/tpf_pbfiles/tpf-fdes/qxhf_parms.gen.dfdl.xsd
3. Service descriptor
/sys/tpf_pbfiles/tpf-fdes/addPassenger.srvc.json
4. REST API documentation
/sys/tpf_pbfiles/tpf-fdes/dfed.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:

/dfed* dfed.swagger.json

API Testing & Management

- Swagger tooling
- API Connect or other tooling for API management

7. Use swagger tooling

Swagger Editor

Swagger UI

Swagger Codegen

<http://swagger.io/tools>

The screenshot shows the Swagger Editor web interface in a browser. The browser's address bar shows the URL `linuxtpf.pok.ibm.com/swagger.editor/`. The interface has a dark grey top bar with a menu (File, Preferences, Generate Server, Generate Client, Help) and a status indicator "All changes saved". The main content area is titled "POST /test" and is divided into two sections: "Parameters" and "Responses".

Parameters

Name	Located in	Description	Required	Schema
<code>prg2op_request</code>	body	request body	Yes	<pre> prg2op_request { source:prg1_input.gen.dfdl.xsd, rootElement:prg1_input, rootInJSON:false prg1_input: ▶ { } } </pre>

Responses

Code	Description	Schema
200	normal response	<pre> prg2op_response_200 { source:prg1_output.gen.dfdl.xsd, rootElement:prg1_output, rootInJSON:false prg1_output: ▶ { } } </pre>

At the bottom of the interface, there is a button labeled "Try this operation".

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



THANK YOU

Questions or comments?

Bradd Kadlecik
z/TPF Development

© 2017 IBM z/TPF | TPF Users Group Spring Conference | IBM Confidential

IBM **z/TPF**
April 5th, 2017

trademarks

IBM, the IBM logo, ibm.com and Rational are trademarks or registered trademarks of International Business Machines Corp., registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on the Web at [“Copyright and trademark information”](#) at www.ibm.com/legal/copytrade.shtml.

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.