REST Message Handlers

Web Services Subcommittee

Bradd Kadlecik

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

IBM Z

IBM

Problem Statement

Common REST provider segments must be created by the application infrastructure to share common processing. These common segments must be updated for every new REST service to determine what processing each service requires and what application program the request needs to be routed to.

Users

Zach
Application developer

How would I go about adding authorization to my REST service?

As-Is User Story

Zach adds the Authorization header to his OpenAPI descriptor and generates the DFDL and C structures by using tpfrestgen. The Authorization header appears as a char * in the C request structure.

OpenAPI header definition:

```
{
  “in” : “header”,
  “name” : “Authorization”,
  “required” : “true”,
  “type” : “string”
}
```

Generated C request structure:

```
struct {
   struct {
      char *Authorization;
   } header;
} request_t;
```

As-Is User Story

Zach creates a routine for handling the authorization but needs to document it internally and make the routine callable by other programs so that others can use it for their REST APIs. The Authorization value will also need to be scrubbed from the structure before continuing.

Pain Points

- There’s no interface to be able to add common processing across multiple REST services.
- Every REST application has access to the authorization credentials and must be trusted to scrub the data.

Value Statement

An application developer can easily orchestrate and manage common routines needed for REST services such as authorization, logging, and AAA management.

To-Be User Story

Zach adds the Authorization header to his OpenAPI descriptor and generates the DFDL and C structures by using tpfrestgen. The Authorization header appears as a char * in the C request structure.

OpenAPI header definition:

```
{
  “in” : “header”,
  “name” : “Authorization”,
  “required” : “true”,
  “type” : “string”
}
```

Generated C request structure:

```
struct {
   struct {
      char *Authorization;
   } header;
} request_t;
```

To-Be User Story

Zach creates a REST message handler to handle the authorization. He can locate the header to process by using a new C function.

z/TPF service descriptor definition:

```
“msgHandlers”:[
  {“program” : “QZZ6”, “type” : “request”}
]
```

QZZ6:

```
tpf_srvc_msgrtn QZZ6(tpf_srvc_msg *msg)
{
  char *authHdr = tpf_httpGetHeader(
            msg->request->headerList,
            msg->request->headerNum,
            “Authorization”);
```

To-Be User Story

Zach now doesn’t want the Authorization header passed to the application, so he removes it from the C structure and changes the DFDL definition to have the Authorization element be defined as document only.

DFDL definition:

<xs:element name=“Authorization" type="xs:string" ~~tddt:indirectKind="pointer" tddt:indirectLength="8" dfdl:lengthKind="delimited" dfdl:lengthUnits="bytes" dfdl:terminator="%NUL;" dfdl:textTrimKind="none"~~ dfdl:inputValueCalc=“{‘’}” ~~default=""~~/>

Users

Zach
Application developer

How do I change how the REST client sees errors from DFDL?

As-Is User Story

Zach decided to use DFDL validation to validate the data for the REST requests.

z/TPF service descriptor definition:

```
“request”: {
  “schema” : “apiRequest_t.gen.dfdl.xsd”,
  “root”: “apiRequest_t”,
  “DFDLValidation” : true
}
```

As-Is User Story

Zach wants to have errors due to data validation returned to the client in the same way that the provider service would. DFDL returns the cause of error in processing the request in the status reason with a status code of 406. He will need to have something modify the response information being returned by z/TPF before it’s sent to the REST client.

Pain Points

- DFDL errors are returned in the status reason of the HTTP response and there’s no way to change it without a user mod.

Value Statement

An application developer can easily modify a DFDL error response while logging the specific error.

To-Be User Story

Zach wants to have errors due to data validation returned to the client in the same way that the provider service would. DFDL returns the cause of error in processing the request in the status reason with a status code of 406.

To-Be User Story

Zach can create a REST message handler to remedy this by creating a JSON body to return the error information with the status reason included.

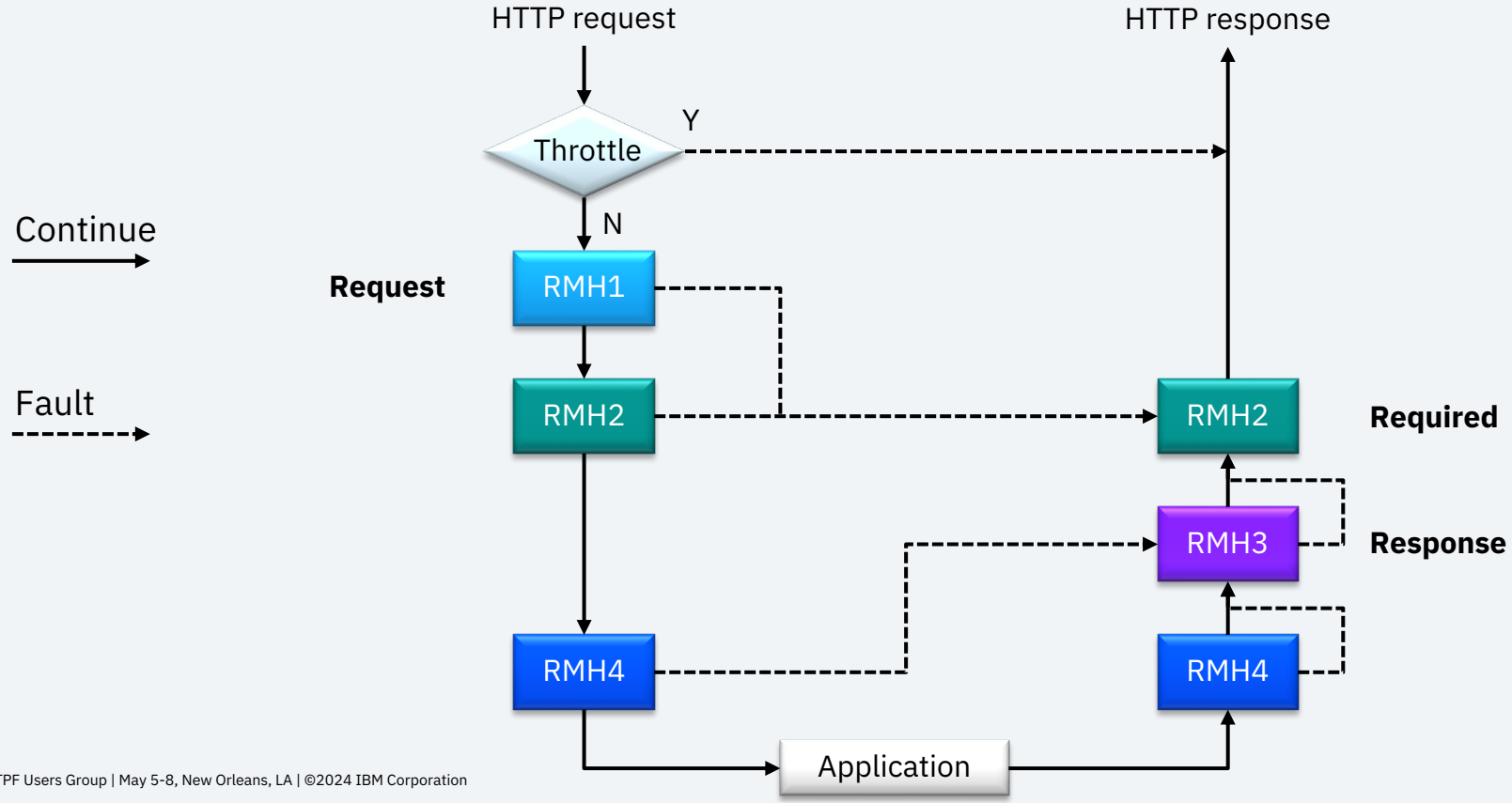
z/TPF service descriptor definition:

```
“msgHandlers”:[
  {“program” : “QZZ7”, “type” : “response”,
“required” : true}
]
```

QZZ7:

```
tpf_srvc_msgrtn QZZ7(tpf_srvc_msg *msg)
{
  if ((msg->action == TPF_SRVC_ACT_PERR) &&
      (msg->errorType == TPF_SRVC_ERR_DFDL)) {
    /* Log error and create JSON body */
    /* modify msg->response for new error return */
    /* return fault to indicate changed response */
    return TPF_SRVC_MSG_FAULT;
  }
```

Technical Details – PJ47150 (Jan 2024)
Overview



Technical Details – PJ47150 (Jan 2024)
z/TPF service descriptor updates

msgHandlers – Contains a list of message handlers to be called

Message Handler object properties:

- **program** – The 4-character program name of the message handler (required).
- **description** – A description of the message handler.
- **required** – Indicates whether the message handler is always to be called for response handling (even for a request message fault).
- **type** – Indicates whether the message handler is to be called for the request, response, or both.
- **userParm** – A 1- to 8-character string value passed to the message handler.

“msgHandlers”:[{“program”:”RMH1”, “type”:”request”}]

Technical Details – PJ47150 (Jan 2024)
REST message handler interface

TPF_SRVC_MSGRTN <PROG> (**tpf_srvc_msg ***);

Return value:
TPF_SRVC_MSG_CONT – Normal return, continue processing
TPF_SRVC_MSG_FAULT – Error return, return the error response provided

tpf_srvc_msg:
action – Indicates if being called for request, response or error by REST consumer or REST provider
msgStatus – The return value from request message handling (or 0 if not called)
errorType – Indicates type of error: fault, timeout, DFDL, interface, service
userParm – The string value specified in the z/TPF service descriptor
request – A pointer to the t_httpClientRequest structure
response – A tpf_httpsvr_resp structure containing the response

Technical Details – PJ47150 (Jan 2024)
ZSRVC DISPLAY update

```
ZSRVC DISPLAY n-servicePostC5
SRVCNAME-servicePostC5    VERSION-1.0.2
  POST /service/serviceCode5
  HOST-http://127.0.0.1:81
  PROXY-NONE
  TIMEOUT-10000      PROVIDERTYPE-Program  PROVIDER-QHH9 _
  UNORDERED-TRUE     DFDLFORMAT-NONE       EXCLUDE-NONE
  MAXREQUESTS-0      MAXREQUESTSERROR-0    MAXREQUESTSWARNINGINTERVAL-0
  PRIORITY-NONE      PRIORITYERROR-0       PRIORITYWARNINGINTERVAL-0
  DFDLVAL-NONE       OASVAL-NONE
  MSGHANDLER  TYPE        USERPARM   REQD    LOADSET          CREATED ON
  RMH1        REQUEST                  N    LOADTPF  09/20/23 17.58.42
  RMH2        ALL               P1     Y       BASE
  RMH3        RESPONSE                 N    LOADTPF  09/20/23 17.58.42
  RMH4        ALL                      N       BASE
```

Conclusion

PJ47150 (Jan 2024): REST message handlers

- An application developer can easily orchestrate and manage common routines needed for REST services such as authorization, logging, and AAA management.
- An application developer can easily modify a DFDL error response while logging the specific error.

Thank you