z/TPF rules engine driver for Java readme

NOTE: Before using this information and the product it supports, read the general information under "Notices" in this document.

CONTENTS

1.0 Introduction

With z/TPF support for Java, you can extend traditional z/TPF applications by using the Java programming language. By using standard Java, Java application programmers can create new services and business logic for your z/TPF system without knowing details of z/TPF programming or how z/TPF operates.

In this programming model, responsibilities are split between the Java and z/TPF application programmers. At a high level, the Java application programmer creates services in Java. Because z/TPF supports standard Java, your Java application programmers can use a wide variety of open source Java packages in addition to writing their own Java code. The services are then packaged in a service application and deployed on the z/TPF system by using the z/TPF application manager for Java (JAM) support.

To use the services written in Java, z/TPF application programmers simply add the tpf_srvcInvoke() function to their application wherever they need to call the new services. When a z/TPF application calls a service that is written in Java, the z/TPF system uses DFDL schemas and generated Java code to automatically convert between C/C++ structures and Java objects. As a result, each side (Java and z/TPF) uses native data constructs without the need to write any conversion code. This programming model provides several benefits:

- The services that are written in Java are implemented by using standard REST interfaces and request and response data is contained in Java objects, so your Java application programmers do not have to know z/TPF.

- Traditional z/TPF applications call services by using a tpf_srvcInvoke() function call, with C/C++ data structures, so your z/TPF application programmers do not have to know Java.

- The z/TPF applications and services are located on the same z/TPF system, so your applications do not incur any communications stack overhead when calling the services.

With JAM support, you can provide highly scalable and available services on the z/TPF system by running multiple IBM J9 Virtual Machines (JVMs) at the same time. Each JVM supports the same set of services and JAM support automatically routes service requests to the next available application thread in the set of JVMs.

The z/TPF rules engine driver for Java demonstrates how you can use Java to incorporate rules engine processing in a z/TPF application and how a rules engine can simplify and add a layer of abstraction to sometimes fragile, messy business logic. The core components of this driver, the flightrules JAM and the flight pricing driver (QRUL), represent a Java rules engine service and a traditional z/TPF application, respectively. These components provide a working example that shows how you can use Java on your z/TPF system and take advantage of Java by calling it from your z/TPF applications.

This document describes how to install, build, and run the QRUL driver and flightrules JAM on your z/TPF system, as well as how to use an optional web application included with this driver.

For more information about creating Java application services and calling those services from your z/TPF applications, see the z/TPF product documentation in IBM Knowledge Center (https://www.ibm.com/support/knowledgecenter/SSB23S).

1.1 Driver components and architecture

The z/TPF rules engine driver for Java contains the following core components, which represent the Java application service and the traditional z/TPF application:

- The priceFlight service is a REST service that is written in Java and uses the open source Drools rules engine to price flights based on origin, destination, loyalty status, seat availability, and other criteria. During rules engine initialization, rules are read from rules files (*.drl files) and stored in rules objects. When the service is started, the service passes the input to the rules engine, which computes a price based on the input and rules objects. The priceFlight service returns the price that is computed by the rules engine to the caller.

  The priceFlight service is packaged in the FlightApp service application and the FlightApp service application is deployed as part of the flightrules JAM.

- The flight pricing driver (QRUL) represents a traditional z/TPF C/C++ application. In this case, the driver creates random sets of pricing criteria (origin, destination, loyalty status, seat availability, and so on) and sends that information as input to the priceFlight service by calling the tpf_srvcInvoke() API. Depending on the driver mode, QRUL displays the results or calls the priceFlight service in a tight loop.

The following components are an optional REST service and web application, which can be used to display the status of the QRUL driver running on your z/TPF system:

- The driverStatus service (QRU3) is a REST service that is written in C++ Language and returns the current status of the flight pricing driver (QRUL).

- A web application is provided that calls the driverStatus service and displays the status of the flight pricing driver in a visual format.

2.0 Change history

2018Mar23 Initial version
2018Oct22 Miscellaneous fixes and enhancements

3.0 Prerequisites

The following list provides the required release levels:

- z/TPF (PUT 14 or later) with z/TPF support for Java (APAR PJ43892) installed. For more information about installing, building, and configuring z/TPF support for Java, see the z/TPF product documentation in IBM Knowledge Center (https://www.ibm.com/support/knowledgecenter/SSB23S).

The following build tools are required:

- maketpf utility

4.0 Installing the driver

1) Use FTP to transfer the tar file (QRUL.tar.gz) to your Linux on IBM Z build system. This file can be placed in any directory as a holding location, for example,

```
/tmp/ztpftar
```

2) Create a root directory to hold the unpacked files, for example, /ztpfdrvs

3) Extract the source code from the tar file by entering the following commands:

```
cd /ztpfdrvs
tar -xvzf /tmp/ztpftar/QRUL.tar.gz
```

The project source files are extracted in the following directory structure:

```
qrul/flightrules.pom.xml
qrul/qru2.cpp
qrul/qru2.mak
qrul/qru3.cpp
qrul/qru3.h
qrul/qru3.mak
qrul/qruj.mak
qrul/qrul.cntl
qrul/qrul.cpp
qrul/qrul.loadfile
qrul/qrul.mak
qrul/qrul_drv.h
qrul/qrul_functions.cpp
qrul/qrul_structures.h
qrul/fdes/ecb_info.gen.dfdl.xsd
qrul/fdes/flightdriver.srvc.json
qrul/fdes/flightdriver.swagger.json
qrul/fdes/flightrules.jam.xml
qrul/fdes/flightrules.srvc.xml
qrul/fdes/flightrules.swagger.xml
qrul/fdes/flightRulesInfo.srvc.json
qrul/fdes/flightRulesInfo_response.gen.dfdl.xsd
qrul/fdes/istream_info.gen.dfdl.xsd
qrul/fdes/status_request.gen.dfdl.xsd
qrul/fdes/status_response.gen.dfdl.xsd
qrul/fdes/ticket_request.gen.dfdl.xsd
qrul/fdes/ticket_response.gen.dfdl.xsd
qrul/maven/dependencies.txt
qrul/rules/base_prices.drl
qrul/rules/rules.drl
qrul/src/main/java/com/flight/app/FlightApp.java
qrul/src/main/java/com/flight/engine/IRulesEngine.java
qrul/src/main/java/com/flight/engine/RulesEngineInstance.java
qrul/src/main/java/com/flight/engine/drools/DroolsEngine.java
qrul/src/main/java/com/flight/engine/drools/DroolsFactory.java
qrul/src/main/java/com/flight/engine/drools/DroolsRuleListener.java
qrul/src/main/java/com/flight/models/CustomerLoyalty.java
qrul/src/main/java/com/flight/models/CustomerType.java
qrul/src/main/java/com/flight/models/FlightRulesInfoResponse.java
qrul/src/main/java/com/flight/models/PriceFlightRequest.java
qrul/src/main/java/com/flight/models/PriceFlightResponse.java
qrul/src/main/java/com/flight/models/SeatSection.java
qrul/src/main/java/com/flight/rest/FlightHandler.java
qrul/src/main/resources/...
qrul/tools/gen_rules.py
qrul/webapp/background.jpg
qrul/webapp/index.html
```

4) Create a maketpf.cfg file with the following contents:

```
APPL_ROOT := /ztpfdrvs
TPF_ROOT := /ztpf
LOADTPF_IP:=ftp://<user>@<host>
```

```
TPF_BSS_NAME := BSS
#TPF_SS_NAME :=
#USER_VERSION_CODE :=
```

a) Set APPL_ROOT to the directory that contains the driver source code that was extracted.
b) Set TPF_ROOT to the directory that contains the z/TPF source code.
c) Set LOADTPF_IP to the correct user/host of your z/TPF system.
d) Set TPF_BSS_NAME to the basic subsystem name of your z/TPF system. By default, this value is set to BSS.
e) Optional: Set TPF_SS_NAME to the subsystem name.
f) Optional: Set USER_VERSION_CODE to any 2-character string. The 2-character string that you set is appended to the shared objects that are built. By default, this value is set to null.

For details about these variables, enter man maketpf.cfg on your Linux on Z build system.

5) Build the USRSTUB program and online program attribute table (IPAT) after you add the QRUL driver control file to your user control file.

a) Add the following line to your user control file (base/cntl/usr.cntl):

```
include qrul/qrul.cntl
```

b) Build the USRSTUB program to generate stubs for all user programs using the following command:

```
maketpf USRSTUB -f
```

c) Rebuild IPAT to incorporate the changes you made in the usr.cntl file:

```
maketpf ipat -f
```

d) Load the IPAT that was built in step 5c to your z/TPF system.

6) If Apache Maven on your Linux on IBM Z build system is configured to use a local repository, verify that all dependency files required by this driver are installed in the local repository and download any missing dependencies. For a list of dependecies required by this driver, see /ztpfdrvs/qrul/maven/dependencies.txt.

7) Run the maketpf utility with the accompanied control file (qrul.cntl) to assemble, compile, and link the driver programs:

```
bldtpf /ztpfdrvs/qrul/qrul.cntl
```

8) Use the standard load procedure to transfer and load the driver shared objects, jar files (Java programs), and common deployment files that are required for the QRUL driver to the z/TPF system:

```
loadtpf -s qrulload /ztpfdrvs/qrul/qrul.cntl  /ztpfdrvs/qrul/qrul.loadfile
```

9) Use the standard procedure to activate these loadsets on the z/TPF system.

10) Update the test driver program to start the QRUL driver.

a) Update base/rt/cvzz.asm (or the tool that runs driver programs) to make an entry for the QRUL driver. The QRUL shared object is the main entry point for the QRUL driver.

b) Build and load the updated CVZZ program to the z/TPF system.

For more information about program management, including how to build and load programs to the z/TPF system, see the z/TPF product documentation in IBM Knowledge Center (https://www.ibm.com/support/knowledgecenter/SSB23S).

5.0 Configuring the z/TPF system

1) To deploy the DFDL schemas, service descriptors, openAPI (swagger) documents, and the JAM descriptor, enter the ZTEST QRUL command with INIT parameter specified. This command calls the ZMDES DEPLOY command for all common deployment files that are required for this driver.

```
   User:   ZTEST QRUL INIT

   System: QRUL0003I 10.27.14 STARTING INIT REQUEST
           CSMP0099I 10.27.14 000000-B ZMDES DEPLOY FILE-FLIGHTRULES.JAM.XML
           MDES0008I 14.58.49 DEPLOY IS COMPLETE ON PROCESSOR B FOR
           FILE-/sys/tpf_pbfiles/tpf-fdes/flightrules.jam.xml
           ...
           QRUL0004I 10.27.14 INIT REQUEST COMPLETE
```

Note: An CSMP0099I and MDES0008I messages will be received for each common deployment file that is in the qrul/fdes source directory.

2) Optional: To enable the flight pricing service to be called from remote systems and to use the web interface to monitor the driver by using a web browser, take the following actions:

a) Update the /etc/tpf_httpserver/url_program_map.conf file on the z/TPF system to include the following entries:

```
      /flightrules/priceFlight      flightrules.swagger.json    2000
      /flightdriver/driverStatus    flightdriver.swagger.json   2000
```

b) After the file in 2a is updated and loaded to the z/TPF system, enter the ZHTPS command to refresh the URL program mapping file.

```
      User:   ZHTPS REF URL

      System: HTPS0030I 15.15.11 HTTP SERVER URL-PROGRAM MAPPING FILE REFRESHED
```

6.0 Starting the flightrules JAM

Enter the ZJAMC START command to start the pricing server in the JAM. For example:

```
    User:   ZJAMC START N-flightrules

    System: JAMC0134I 13.31.25 THE JVM FOR JAM flightrules IS STARTED, PID 1092157451.
            JAMC0002I 13.31.25 JAM flightrules IS ACTIVE.
```

7.0 Running the QRUL driver

Before you start the driver, ensure that the pricing server is running by entering the ZJAMC DISPLAY command. For example:

```
    User:   ZJAMC DISPLAY N-flightrules

    System: JAMC0007I 13.48.07 START OF ZJAMC DETAILED DISPLAY
            JAM NAME          #JVMS #THDS SHARED CLASS CACHE        STATE
            flightrules           1     4 N/A                       ACTIVE

            DEPLOYMENT DESCRIPTOR FILE NAME
            /sys/tpf_pbfiles/tpf-fdes/flightrules.jam.xml

            JVM PID      JVM STATE       JVM LOG FILE DIRECTORY
            1092157451   ACTIVE          /tpfjam/flightrules/1092157451
            END OF DISPLAY+
```

1) Start the driver in one of the following ways:

o To start the driver in visual mode, enter ZTEST QRUL START. Every few seconds a flight pricing request is be generated and the response is displayed on the console. For example:

```
      User:   ZTEST QRUL START

      System: QRUL0001I 10.29.49 THE QRUL DRIVER IS STARTED IN VISUAL MODE
```

Example console output when running in visual mode:

```
  QRUL0011I 11.22.05 REQUEST AND RESPONSE DATA FOR THE priceFlight SERVICE
  ------------------------------------------------------------------------
  Request: {PHL to AVP (531 miles), Sun - 02/17/19, Free Seats: 70/250,
            3 bag(s), ONE WAY, BUSINESS, PLATINUM, SENIOR}
  Response:
    Price: $158
    Rules:
     -Business class tickets have a base price of $95
     -Mileage costs based on loyalty: Silver $.15 | Gold $.12 | Platinum $.08
     -Tickets are $10 more if 20% - 50% of seats are available
     -Platinum customers receive 2 free checked bags; additional bags cost $20
     -Senior citizens receive a 5% discount
  ------------------------------------------------------------------------
  END OF DISPLAY
```

- To start the driver in load test mode, enter ZTEST QRUL START mode-1. In load test mode, request and response information is not displayed automatically and requests to the priceFlight serivce are made in a tight loop. For example:

```
User:   ZTEST QRUL START MODE-1

System: QRUL0001I 10.25.02 THE QRUL DRIVER IS STARTED IN LOAD MODE
```

2) To display a summary of the driver status, enter ZTEST QRUL STATUS. For example:

```
User:   ZTEST QRUL STATUS

System: QRUL0010I 10.30.02 QRUL DRIVER STATUS
        ****************************************************************************
        IS#   ECB#   #200/400/500/FAIL       RATE      MIN/MAX(ms)   MEAN/STDDEV(ms)
        ----------------------------------------------------------------------------
         1       0       2113741/0/0/0     7080/sec    0.106/ 8.977   0.139/ 0.195
        ----------------------------------------------------------------------------
          1 ECB(s)       2113740/0/0/0     7080/sec    0.106/ 8.977   0.139/ 0.195
        ----------------------------------------------------------------------------
        Throttle:         0 us                                          Mode:   LOAD
        Time Running: 00:05:00                                         Rules:     15
        ****************************************************************************
        END OF DISPLAY
```

3) To stop the driver, enter ZTEST QRUL STOP. For example:

```
User:   ZTEST QRUL STOP

System: QRUL0007I 10.30.08 STOPPING THE QRUL DRIVER
        QRUL0008I 10.30.08 THE QRUL DRIVER IS STOPPED
```

For information about additional parameters for the QRUL driver command, including how to run multiple QRUL driver ECBs and how to add a delay between service requests, enter ZTEST QRUL HELP.

8.0 Running the driver status web application

To display the driver status, you can enter ZTEST QRUL STATUS. Additionally, you can display the driver status by using a web browser. A simple web graphical user interface (GUI) is provided in the /ztpfdrvs/qrul/webapp/ directory.

1) Open /ztpfdrvs/qrul/webapp/index.html in a web browser.

2) In the Host field, enter the IP address or hostname of the z/TPF system.

3) In the Port field, enter the port number for the HTTP server.

4) Click Get Status. I-stream and ECB information for the flight pricing driver is displayed.

9.0 Optional driver adjustments

1) By default, the flightRules JAM is configured to start one JVM with four application threads and is able to run with a maximum of 256 1-MB frames for 64-bit heap; that is, a value of 256 for the MAXXMMES parameter in keypoint A. These are minimal settings that demonstrate how a traditional z/TPF C/C++ application can call Java services on z/TPF. The following are some settings that can be changed to provide improved application performance and scalability.

   If you make these changes, ensure that there are enough 1 MB frames allocated to accommodate each JVM that uses the amount of 64-bit ECB heap defined by the MAXXMMES parameter.

   o Optional: To run the flightRules JAM with more JVMs or application threads, change the value of the <NumberJVMs> or <NumberThreadsPerJVM> elements in the /ztpfdrvs/qrul/fdes/flightrules.jam.xml file and load the updated flightRules JAM descriptor to the z/TPF system.

   o Optional: To provide more ECB heap to the JVMs, increase the MAXXMMES parameter value in keypoint A to 600 MB or greater.

2) This driver includes a small number of rules for use by the rules engine. You can add additional rules by creating new rules files and loading them to /sys/tpf_pbfiles/apps/flightrules/rules directory on your z/TPF system and recycling the flightrules JAM. You can create new rules files manually or use the script, /ztpfdrvs/qrul/tools/gen_rules.py, to generate rules.

10.0 Notices

This information was developed for products and services offered in the US.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not grant you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive, MD-NC119
Armonk, NY 10504-1785
US

For license inquiries regarding double-byte character set (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

Intellectual Property Licensing
Legal and Intellectual Property Law
IBM Japan Ltd.
19-21, Nihonbashi-Hakozakicho, Chuo-ku
Tokyo 103-8510, Japan

INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some jurisdictions do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM websites are provided for convenience only and do not in any manner serve as an endorsement of those websites. The materials at those websites are not part of the materials for this IBM product and use of those websites is at your own risk.

IBM may use or distribute any of the information you provide in any way it believes appropriate without incurring any obligation to you.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

IBM Director of Licensing
IBM Corporation
North Castle Drive, MD-NC119
Armonk, NY 10504-1785
US

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

10.1 Trademarks

IBM, the IBM logo, and ibm.com are 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.

Linux is a registered trademark of Linus Torvalds in the United States, other countries, or both.

Java and all Java-based trademarks are trademarks or registered trademarks of Oracle and/or its affiliates.

10.2 Warranty

This package is provided on an "as is" basis. There are no warranties, express or implied, including the implied warranties of merchantability and fitness for a particular purpose. IBM has no obligation to provide service, defect correction, or any maintenance for the package. IBM has no obligation to supply any updates or enhancements for the package to you even if such are or later become available.