z/TPF stateful hello world readme

CONTENTS

1.0 Introduction

This sample driver code demonstrates how to utilize stateful services in a Java application. To use stateful service requests, you must set the providerType element to StatefulProgram in the service descriptor. Two examples are included on this package:

1) An application that makes one or more requests to update a counter.
   o Make a stateful request from a Java application (HelloWorldSF.java) to a stateful z/TPF application service provider - QRSF.CPP

   o QRSF.CPP increments a global variable that holds the counter value and returns the value to the Java application (HelloWorldSF.java) by calling the tpf_srvcSendResponse() API.

   Note: The Java application (HelloWorldSF.java) makes n requests in a single thread based on the returned counter value.

2) A single thread to intersperse the following items:
   o A stateful service request to open, read from, and hold a z/TPFDF subfile. The service returns the open z/TPFDF subfile to the Java client.

   o A stateful service request for z/TPFDF to add or replace the sample reservation and to unhold and close the z/TPFDF subfile.

1.1 Driver components and architecture

The z/TPF stateful hello world sample code contains the following core components, which represent the Java application and the traditional z/TPF application:

o HelloWorldSF.java is a Java application that makes stateful requests.
   The HelloWorldSF.java application, and all of the associated swagger documents and service descriptors are packaged in the QRSF jar file.

o Counter sample driver
   The counter service (qrsf.cpp) is a REST service that is written in C++ Language.
   This service keeps a counter and increments the counter with every service request.
   For the last stateful service request, QRSF returns the final count value to the Java application (HelloWorldSF.java).

o z/TPFDF sample driver
   * qrs1.cpp is a REST service that is written in C++ Language. This service opens and reads from a z/TPFDF subfile and holds the record for a flight reservation. The

```
     service returns a z/TPFDF file pointer to the Java application (HelloWorldSF.java).

   * qrs2.cpp is a REST service that is written in C++ Language. This service adds or
     replaces, unholds, and closes the previous z/TPFDF file pointer.
```

2.0 Change history

```
2018Dec21  Initial version
2019Jan30  Minor housekeeping updates
```

3.0 Prerequisites

```
The following list provides the required release levels:
  o z/TPF with APAR PJ45433 and all prerequisite APARs applied. For more information,
    see the apedit for PJ45433. 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).
```

4.0 Installing the driver

```
1) Use FTP to transfer the tar file (QRSF.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/QRSF.tar.gz

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

   List of files:
   -------------
   o ./qrsf
     * c_dr21fm.h
     * dford.mac
     * dr21fm.mac
     * _flight.h
     * init_flight.cpp
     * qrsf.cpp
     * qrs1.cpp
     * qrs2.cpp
     * qrs1.mak
     * qrs2.mak
     * qrsf.cntl
     * qrsf.h
     * qrsf.load
     * qrsf.loadfile
     * qrsf.mak
     * qrsf_maven.mak
     * qrsf_pom.xml
     * uf8s.asm

   o ./fdes:
     * dfhelloworld1.srvc.json  # service provider type = "StatefulProgram"
     * dfhelloworld2.srvc.json  # service provider type = "StatefulProgram"
     * sfhelloworld.srvc.json   # service provider type = "StatefulProgram"

     * inputSamp.gen.dfdl.xsd
     * outputSamp.gen.dfdl.xsd
```

```
      * qrsfParmsReq.gen.dfdl.xsd

      * sfhelloworld.swagger.json

   o ./maven/dependencies.txt
   o ./src/main/main/java/com/ibm/tpf/qrsf/HelloWorldSF.java

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
   QRSF driver control file to your user control file.

   a) Add the following line to your user control file (base/cntl/usr.cntl):
       include qrsf/qrsf.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/qrsf/maven/dependencies.txt.

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

      bldtpf /ztpfdrvs/qrsf/qrsf.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 QRSF driver
   to the z/TPF system:

      loadtpf -s qrsfload /ztpfdrvs/qrsf/qrsf.cntl  /ztpfdrvs/qrsf/qrsf.loadfile

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

Complete the following steps (10 - 13) only for the z/TPFDF portion of the driver:

10) Add the DBDEF program UF8S to your local_mod version of tpfdf/macro/dfuex.mac.
    This change will be incorporated when UFC8 is rebuilt in the next step.
    For example:
    GETPC NAME=UF8S,LOCK=YES,ADDR=R2   LOCK PROGRAM IN CORE
    ENTRC UF8S                          QRSF DRIVER DBDEFS
    BR R7
```

11) Build and load the updated UFC8 programs to the z/TPF system.

12) Enter ZUDFM DEF INIT to rebuild the DBDEF index table.

13) Enter the following command to initialize the database and create the predefined data in the database. Enter the command twice.

```
    ZUDFM INIT BB21
```

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

Deploy the openAPI (swagger) documents by entering the ZMDES command. For example:

```
User:   zmdes deploy file-sfhelloworld.swagger.json

System: CSMP0097I 14.17.05 CPU-B SS-BSS  SSU-HPN  IS-01
        MDES0008I 14.17.05 DEPLOY IS COMPLETE ON PROCESSOR B FOR
        FILE-/sys/tpf_pbfiles/tpf-fdes/sfhelloworld.swagger.json+
```

6.0 Running the z/TPF stateful hello world sample driver

By default, the jar files are deployed in the /sys/tpf_pbfiles/apps/qrsf directory on the z/TPF system. Before you run the driver, change your working directory to the directory that contains the jar files. For example:

```
   zfile cd /sys/tpf_pbfiles/apps/qrsf
```

o To run the counter sample driver, enter the ZFILE JAVA command and specify the count value. For example:

```
  User:   zfile java -Xjit -cp .:*:/sys/tpf_pbfiles/apps/tpfjax/tpfclient.jar
          com/ibm/tpf/HelloWorldSF count 3

  System: Nov 14, 2018 2:20:36 PM com.ibm.tpf.HelloWorldSF CounterSample
          INFO: Java Client Counter = 1 _
          Nov 14, 2018 2:20:36 PM com.ibm.tpf.HelloWorldSF CounterSample
          INFO: Java Client Counter = 2
          Nov 14, 2018 2:20:36 PM com.ibm.tpf.HelloWorldSF CounterSample
          INFO: Java Client Counter = 3
          END OF DISPLAY+
```

o To run the z/TPFDF sample driver, enter the ZFILE JAVA command and specify df. The requests are filled in randomly. For example:

```
  User:   zfile java -Xjit -cp .:*:/sys/tpf_pbfiles/apps/tpfjax/tpfclient.jar
          com/ibm/tpf/HelloWorldSF df

  System: Nov 14, 2018 3:31:21 PM com.ibm.tpf.HelloWorldSF DFSample
          INFO: Response :class QrsfParmsReq { _
           qrsfParmsReq: class QrsfParmsReqQrsfParmsReq {
               fltNum: 177
               fltBrd: 101
               fltDest: 156
               fltAtype: 954
               fltTime: 12:00
               fltAvail: 999
               filePtr: 593269504
```

```
                }
           }
          Nov 14, 2018 3:31:21 PM com.ibm.tpf.HelloWorldSF DFSample
          INFO: Request :class FlightAdd {
                fltNum: 133 _
                fltBrd: 127
                fltDest: 146
                fltAtype: 954
                filePtr: 593269504
          }
          END OF DISPLAY+
```

7.0 Notices

7.1 Trademarks

7.2 Warranty