
z/OS batch in Java

Running Java in batch can be an interesting solution for some applications. It does not have to be a heavy-duty batch job that shuffles around millions of records of data. Instead, it could be useful in cases where you want to reuse Java Beans with business or validation logic to perform certain tasks at a scheduled time.

Running a Java application as a batch job

The z/OS utility BPXBATCH runs an arbitrary z/OS UNIX shell command or executable. You can use it to run Java applications in a batch environment.

The following example is a complete JCL for running a Java program in batch.

```
//NOLTING0 JOB '','NOLTING LS-D115',CLASS=A,MSGCLASS=H,MSGLEVEL=(1,1),
// REGION=4096K,TIME=(5,0),NOTIFY=&SYSUID
//*****
//* run a java program in batch mode
//*****
//STEP1 EXEC PGM=BPXBATCH,
// PARM='SH java Test1'
//STDOUT DD PATH='/u/nolting/stdout',
// PATHOPTS=(OWRONLY,OCREAT,OTRUNC),
// PATHMODE=SIRWXU
//STDERR DD PATH='/u/nolting/stderr',
// PATHOPTS=(OWRONLY,OCREAT,OTRUNC),
// PATHMODE=SIRWXU
//STDENV DD *
CLASSPATH=/u/nolting:$CLASSPATH
/*
```

The BPXBATCH default for stdin and stdout is /dev/null. If you do not specify anything for stderr, it will use to the same definition as stdin.

Sometimes you might want to control the batch job's output in a familiar environment, such as SDSF. In this case you can add an extra step to redirect the output to an SYSOUT=* DD card, as shown in the following example:

```

//NOLTING0 JOB ' ', 'NOLTING LS-D115', CLASS=A, MSGCLASS=X, MSGLEVEL=(1,1),
// REGION=4096K, TIME=(5,0), NOTIFY=&SYSUID
//*****
//* run a java program in batch mode
//*****
//STEP1 EXEC PGM=BXPBATCH,
// PARM='SH java itso.jni.CB12C2JWineReader'
//*-----
//STDIN DD DUMMY
//STDENV DD DUMMY
//STDOUT DD PATH='/tmp/&SYSUID..out',
// PATHOPTS=(OWRONLY,OCREAT,OTRUNC),
// PATHMODE=SIRWXU
//STDERR DD PATH='/tmp/&SYSUID..err',
// PATHOPTS=(OWRONLY,OCREAT,OTRUNC),
// PATHMODE=SIRWXU
//*****
//STEP1 EXEC PGM=IKJEFT01 , DYNAMNBR=300, COND=EVEN
//*-----
//SYSPRINT DD SYSOUT=*
//HFSOUT DD PATH='/tmp/&SYSUID..out'
//HFSERR DD PATH='/tmp/&SYSUID..err'
//STDOUTL DD SYSOUT=*, DCB=(RECFM=VB, LRECL=133, BLKSIZE=137)
//STDERRL DD SYSOUT=*, DCB=(RECFM=VB, LRECL=133, BLKSIZE=137)
//SYSPPRINT DD SYSOUT=*
//SYSTSIN DD *
OCOPY INDD(HFSOUT) OUTDD(STDOUTL)
OCOPY INDD(HFSERR) OUTDD(STDERRL)
/*

```

For further information about the BXPBATCH utility and how to run UNIX programs in batch, refer to *z/OS UNIX System Services User's Guide*, SA22-7801.

Defining environment variables

You probably need to do some environment setup for your Java application. Usually, you set the environment variables in an environment variable file. The file is identified with a JCL STDENV DD statement, and can be either an HFS file or an z/OS data set.

For example:

```
STDENV DD PATH='u/nolting/env.file', PATHOPTS=ORDONLY
```

The file must be a text file specified with one variable per line in the format variable=value. Environment variable names must begin in column 1. Here is an example for an environment file that sets up Java-related environment variables:

```

JAVA_HOME=/usr/lpp/java18/J1.1
PATH=/usr/lpp/java18/J1.1/bin:$PATH
CLASSPATH=$CLASSPATH:/usr/lpp/IBMDebug/lib/derdbg1.jar:/usr/lpp/IBMDebug
/lib/derdebug.jar

```

If you are using an z/OS data set, it must be a sequential data set, a partitioned data set (PDS) member, or a SYSIN data set. Record format can be fixed or variable (non-spanned). Make sure that you have NUMBER OFF before editing the file, as sequence numbers in the file will cause an error.

You can also specify the environment variables as part of an in-stream JCL SYSIN data set. For example:

```
//STDENV DD *
JAVA_HOME=/usr/lpp/java18/J1.1
PATH=/usr/lpp/java18/J1.1/bin:$PATH
CLASSPATH=/u/nolting:$CLASSPATH
/*
```

Another alternative is to add the variables to the UNIX System Services .profile file belonging to the user ID that is running the batch job.

Specifying the Java application to run

You specify the Java application as a parameter to the BPXBATCH program. In the following example we show how you specify the Test1 application to run:

```
//STEP1 EXEC PGM=BPXBATCH,
// PARM='SH java Test1'
```

The stdout and stderr output will go to the files specified in the JCL STDOUT and STDERR DD statements.

You might also want to create a shell script that calls the Java application. The JCL to run a shell script would look like this:

```
//STEP1 EXEC PGM=BPXBATCH,
// PARM='SH shtest1'
```

This is particularly useful in cases where you have to specify long Java properties. All the definitions could be made in the shell script, as shown in the following example:

```
CLASSPATH=/u/nolting:$CLASSPATH
java -Djava.naming.factory.initial=com.sun.jndi.cosnaming.CNCTXFactory
-Djava.naming.provider.url=iiop://9.12.2.26:1100 hello.RemoteHelloServer
```

Running a Java application as an z/OS started task

It is also possible to run a Java application as an z/OS started task (STC). This could be useful for starting Java servers, for example an RMI server. Note that the Java application runs under the authorization of the STC user ID, which allows you to assign specific authorities to a Java server application.

In addition, your Java server runs in a familiar, "operator-friendly" environment and can be easily started, monitored and cancelled from an z/OS console.

We recommend that the STC user is a valid OMVS user with a .profile file. Environment variables should be placed in its .profile file.

As illustrated in the following example, the JCL for the STC is more or less the same as for the batch job:

```
//JSERVER PROC
//STEP1 EXEC PGM=BPXBATCH,
// PARM='SH java itso.jbatch.RMISampleServer'
//STDOUT DD PATH='/u/nolting/stdout1',
//  PATHOPTS=(OWRONLY,OCREAT,OTRUNC),
//  PATHMODE=SIRWXU
//STDERR DD PATH='/u/nolting/stderr1',
//  PATHOPTS=(OWRONLY,OCREAT,OTRUNC),
//  PATHMODE=SIRWXU
```