Getting Started Tutorial

REVISION HISTORY			
NUMBER	DATE	DESCRIPTION	NAME
1.0	2019-09	Initial release	MV

Contents

Chapter 1

About the guide

This document guides user how to quickly start using Enduro/X for Java module. It contains a step by step actions what needs to be taken to write and start the basic XATMI server and client process

Chapter 2

Operating System configuration

Certain operating system configuration is required in order to Enduro/X running, see ex_adminman(guides)(Enduro/X Administration Manual, Setup System) section, the steps are crucial to be executed for chosen operating system before continuing.

Chapter 3

Installing binary packages, for Ubuntu

After the system configuration is done, Enduro/X and Java packages can be installed. This includes installation of JDK and build essentials, because this sample will also show how to use the Enduro/X Java linker tool to produce the standard executables out from jar files (exact package names download from endurox.org)

```
$ sudo -s
# sudo apt install default-jdk build-essential
# dpkg -i endurox-7.0.3-1.linuxmint18_3_GNU_epoll.x86_64_64.deb
# dpkg -i endurox-java-1.0.1-1.linuxmint18_3_java1_8.x86_64.deb
```

As our tutorial is showing how to link the Java binaries in to standard executables, following two directories needs to be known:

- Directory where **libjava.so** lives;
- Directory where **libjvm.so** lives;

You may find these folder by using "find" tool, e.g.:

```
$ cd /usr
$ find . -name libjava.so
$ find . -name libjvm.so
```

Or try to resolve which java version is used:

```
$ which java
/usr/bin/java

$ ls -l /usr/bin/java
lrwxrwxrwx 1 root root 22 Apr  7  2018 /usr/bin/java -> /etc/alternatives/java

$ ls -l /etc/alternatives/java
lrwxrwxrwx 1 root root 46 Apr  7  2018 /etc/alternatives/java -> /usr/lib/jvm/java-8- ↩
    openjdk-amd64/jre/bin/java


$ cd /usr/lib/jvm/java-8-openjdk-amd64

$ find . -name libjava.so
./jre/lib/amd64/libjava.so

$ find . -name libjvm.so
./jre/lib/amd64/server/libjvm.so
```

So following folders has been extracted:

- /usr/lib/jvm/java-8-openjdk-amd64/jre/lib/amd64 (for libjava.so)
- /usr/lib/jvm/java-8-openjdk-amd64/jre/lib/amd64/server (for libjvm.so)

Chapter 4

Configuring the application environment

We will run our app "app-java" from new user "user1". Application domain will be located in /opt/app-java folder. The initial structure is used to hold the directory structure of the sources and runtime.

To create creating base environment, Enduro/X "provision" tool is used. This generates base layout of application. Additional folder will be created later.

```
# sudo -s
# useradd -m user1
# mkdir -p /opt/app-java
# chown user1 /opt/app-java
# su - user1
$ cd /opt/app-java
$ xadmin provision -d \
    -v qprefix=appj \
    -v installQ=n  \
    -v eventSv=n \
    -v cpmSv=n \
    -v configSv=n \
    -v bridge=n \
    -v addubf=test.fd
....
Provision succeed!
```

Above script uses defaults, and for some parameters values are specified:

Table 4.1: Parameters applied to provision script above

Parameter	**Value applied**	**Description**
qprefix	app-java	Application prefix
installQ	n	Do not configure persistent queue as not used in example
eventSv	n	Do not configure event server as not used in example

Table 4.1: (continued)

Parameter	Value applied	Description
cpmSv	n	Do not configure client process monitor server as not used in example
configSv	n	Do not configure Common Configuration interface server as not used here
bridge	n	Do not install network bridge as not used in example
addubf	test.fd	Additional Key-value field table/UBF field to be configured, used by sample later

After provision completed, add directories for source code

```
$ mkdir /opt/app-java/test
$ mkdir -p /opt/app-java/src/testcl
$ mkdir -p /opt/app-java/src/testsv
```

Thus the final directory structure built for the application is

- /opt/app-java/conf - will contain configuration files.
- /opt/app-java/conf/setappj - environment configuration file.
- /opt/app-java/conf/ndrxconfig.xml - XATMI server process configuration file.
- /opt/app-java/conf/app.ini - application settings.
- /opt/app-java/src/testcl - Enduro/X sample client process source
- /opt/app-java/src/testsv - Enduro/X sample server process sources.
- /opt/app-java/bin - for executables.
- /opt/app-java/ubftab - for tables for field definitions.
- /opt/app-java/tmp - temp dir

- /opt/app-java/log - for logging
- /opt/app-java/test - test data

For demo purposes the provision script have made more or less empty XATMI server configuration file found in **/opt/app-java/conf/ndrxconfig.xml**. Lets register firstly our XATMI server named **testsv** here first. Do this in **<servers/>** section add following **<server />** block in the file:

```
<?xml version="1.0" ?>
<endurox>
...
        </defaults>
        <servers>
        <server name="testsv">
            <min>1</min>
            <max>1</max>
            <srvid>20</srvid>
            <sysopt>-e ${NDRX_APPHOME}/log/testsv.log -r</sysopt>
        </server>
        <server name="testsv">
            <srvid>120</srvid>
            <min>1</min>
            <max>1</max>
            <sysopt>-e ${NDRX_APPHOME}/log/java.log -r</sysopt>
            <cmdline>java -cp ${CLASSPATH}:${NDRX_APPHOME}/src/testsv/testsv.jar Testsv</ ↩
                cmdline>
        </server>
        </servers>
</endurox>
```

We have done adding the server process to XML config in two ways. First configuration of java server process **testsv** is done by booting it as linked with Enduro/X Java Linker tool **exjld(8)**, and the second testsv is booted as pure java process.

Additional environmental configuration shall be made for the application, as it needs the class path for the endurox.jar binding classes and also for linked processes library paths of shared libs are required.

Thus append the **/opt/app-java/conf/setappj** with:

```
...
export CLASSPATH=/usr/share/java/enduroxjava.jar
export LD_LIBRARY_PATH=/usr/lib/jvm/java-8-openjdk-amd64/jre/lib/amd64:/usr/lib/jvm/java-8- ↩
    openjdk-amd64/jre/lib/amd64/server

#
# For MacOS set to something like this:
#
#export DYLD_LIBRARY_PATH=$DYLD_LIBRARY_PATH:/System/Library/Frameworks/ImageIO.framework/ ↩
    Versions/A/Resources:$JAVA_HOME/jre/lib
#export DYLD_LIBRARY_PATH=$DYLD_LIBRARY_PATH:$JAVA_HOME/jre/lib/server
```

Also we are about to add some logging settings for our binaries we are about to build, thus add following lines int the **[@debug]** section in application ini file (**/opt/app-java/conf/app.ini**):

```
[@debug]
..
testsv= ndrx=2 ubf=0 tp=5 file=
testcl= ndrx=2 ubf=0 tp=5 file=
java= ndrx=2 ubf=0 tp=5 file=
```

To learn more about debug configuration, see ndrxdebug.conf(5) manpage, note that document describes both formats legacy, where separate file was used and current one with Common-Configuration (i.e. using ini file section).

Chapter 5

Adding source code

This tutorial will use Enduro/X built in feature of generating the source code. Generators can create simple XATMI client/server processes. IN this example, there will create a call from client to server where client process adds one field to UBF (key/value buffer) and the server process will return one field.

The development process will be following:

- Define UBF tables and compile them for java
- Create Java server process
- Create Java client process

5.1 Define UBF tables

Enduro/X **Exfields** table in folder **/opt/app-java/ubftab** is already added by provision command. In this folder we will add additional **test.fd** (already included in configuration). Also will provide make file to generate **ubftab.jar** for java processes. Then folder will be provided as symlink to src folder as generated sources seeks for UBF table package one level up.

```
$ cd /opt/app-java/ubftab

$ xadmin gen ubf tab 2>/dev/null
 0: table_name   :UBF Table name (.fd will be added) [test]:
 1: base_number  :Base number [6000]:
 2: testfields   :Add test fields [y]:
 3: genexfields  :Gen Exfields [y]:
 4: genmake      :Gen makefile [y]:
 5: makeLang     :Target language (c/go/java) [c]: java

*** Review & edit configuration ***

 0: Edit table_name   :UBF Table name (.fd will be added) [test]:
 1: Edit base_number  :Base number [6000]:
 2: Edit testfields   :Add test fields [y]:
 3: Edit genexfields  :Gen Exfields [y]:
 4: Edit genmake      :Gen makefile [y]:
 5: Edit makeLang     :Target language (c/go/java) [java]:
c: Cancel
w: Accept, write
Enter the choice [0-6, c, w]: w
Gen ok!

$ make -f Mjava 2>/dev/null
```

```
$SOURCES is [test.fd Exfields]
$CLASS is [test.class Exfields.class]
$OUTPUT is [test.java Exfields.java]
$FIELDTBLS is [test.fd,Exfields]
mkfldhdr -m2 test.fd Exfields
javac test.java
javac Exfields.java
jar -cmf manifest.txt ubftab.jar test.class Exfields.class
rm Exfields.java test.java

$ ls -l
total 64
-rw-rw-r-- 1 user1 user1 14952 Sep  8 13:17 Exfields
-rw-rw-r-- 1 user1 user1  5060 Sep  8 13:17 Exfields.class
-rw-rw-r-- 1 user1 user1   189 Sep  8 13:17 Makefile
-rw-rw-r-- 1 user1 user1   812 Sep  8 13:17 Mjava
-rw-rw-r-- 1 user1 user1  1126 Sep  8 13:17 test.class
-rw-rw-r-- 1 user1 user1  1301 Sep  8 13:17 test.fd
-rw-rw-r-- 1 user1 user1  3099 Sep  8 13:17 ubftab.jar
-rw-rw-r-- 1 user1 user1 18890 Sep  8 13:17 ULOG.20190908
```

At certain points Enduro/X will provide extra logs, as environment is not yet configured (at-least when doing generate), thus we redirect the logs to **2>/dev/null**.

At the end we see that **ubftab.jar** is generated. To see the actual source code for the "Exfields.class" or "test.class", change the **Mjava** and include following line

```
.PRECIOUS: Exfields.java test.java
```

And run make again:

```
$ make -f Mjava clean
$ make -f Mjava
$ ls -l *.java
-rw-rw-r-- 1 user1 user1 13048 Sep  8 13:27 Exfields.java
-rw-rw-r-- 1 user1 user1  2225 Sep  8 13:27 test.java
```

Thus test.java contains field definition id constants:

```
public final class test
{
          /** number: 6011 type: char*/
          public final static int T_CHAR_FLD = 67114875;
          /** number: 6012 type: char*/
          public final static int T_CHAR_2_FLD = 67114876;
          /** number: 6021 type: short*/
          public final static int T_SHORT_FLD = 6021;
          /** number: 6022 type: short*/
          public final static int T_SHORT_2_FLD = 6022;
          /** number: 6031 type: long*/
          public final static int T_LONG_FLD = 33560463;
          /** number: 6032 type: long*/
          public final static int T_LONG_2_FLD = 33560464;
          /** number: 6041 type: float*/
          public final static int T_FLOAT_FLD = 100669337;
          /** number: 6042 type: float*/
          public final static int T_FLOAT_2_FLD = 100669338;
          /** number: 6043 type: float*/
          public final static int T_FLOAT_3_FLD = 100669339;
          /** number: 6051 type: double*/
          public final static int T_DOUBLE_FLD = 134223779;
          /** number: 6052 type: double*/
```

```
        public final static int T_DOUBLE_2_FLD = 134223780;
        /** number: 6053 type: double*/
        public final static int T_DOUBLE_3_FLD = 134223781;
        /** number: 6054 type: double*/
        public final static int T_DOUBLE_4_FLD = 134223782;
        /** number: 6061 type: string*/
        public final static int T_STRING_FLD = 167778221;
        /** number: 6062 type: string*/
        public final static int T_STRING_2_FLD = 167778222;
        /** number: 6063 type: string*/
        public final static int T_STRING_3_FLD = 167778223;
        /** number: 6064 type: string*/
        public final static int T_STRING_4_FLD = 167778224;
        /** number: 6065 type: string*/
        public final static int T_STRING_5_FLD = 167778225;
        /** number: 6066 type: string*/
        public final static int T_STRING_6_FLD = 167778226;
        /** number: 6067 type: string*/
        public final static int T_STRING_7_FLD = 167778227;
        /** number: 6068 type: string*/
        public final static int T_STRING_8_FLD = 167778228;
        /** number: 6069 type: string*/
        public final static int T_STRING_9_FLD = 167778229;
        /** number: 6010 type: string*/
        public final static int T_STRING_10_FLD = 167778170;
        /** number: 6081 type: carray*/
        public final static int T_CARRAY_FLD = 201332673;
        /** number: 6082 type: carray*/
        public final static int T_CARRAY_2_FLD = 201332674;
}
```

If more advanced packages of UBF tables are needed, then see **mkfldhdr(8)** manpage, i.e. -p parameter.

5.2 Define Java server process

Lets generate simple Java server process which provides some field in responses of the service calls. We will do this by using Enduro/X generators:

```
$ cd /opt/app-java/src/testsv
$ xadmin gen java server 2>/dev/null
 0: classname    :Java class name of XATMI Server [Testsv]:
 1: pkgname      :Java package name [testsv.jar]:
 2: svcname      :Service name [TESTSV]:
 3: useubf       :Use UBF? [y]:
 4: ubfname      :UBF package name [ubftab.jar]:
 5: config       :INI File section (optional, will read config if set) []:
 6: genmake      :Gen makefile [y]:
 7: enduroxjar   :Full path (incl filename) to enduroxjava.jar [/usr/share/java/enduroxjava ↩
    .jar]:
 8: makebin      :Link as binary [y]:
 9: binname      :Binary name [testsv]:
10: libpath_java :Lib path for libjava [/usr/lib/jvm/java-8-openjdk-amd64/jre/lib/amd64]:
11: libpath_jvm  :Lib path for libjvm [/usr/lib/jvm/java-8-openjdk-amd64/jre/lib/amd64/ ↩
    server]:

*** Review & edit configuration ***

 0: Edit classname    :Java class name of XATMI Server [Testsv]:
 1: Edit pkgname      :Java package name [testsv.jar]:
 2: Edit svcname      :Service name [TESTSV]:
```

```
 3: Edit useubf       :Use UBF? [y]:
 4: Edit ubfname      :UBF package name [ubftab.jar]:
 5: Edit config       :INI File section (optional, will read config if set) []:
 6: Edit genmake      :Gen makefile [y]:
 7: Edit enduroxjar   :Full path (incl filename) to enduroxjava.jar [/usr/share/java/ ↩
    enduroxjava.jar]:
 8: Edit makebin      :Link as binary [y]:
 9: Edit binname      :Binary name [testsv]:
10: Edit libpath_java :Lib path for libjava [/usr/lib/jvm/java-8-openjdk-amd64/jre/lib/ ↩
    amd64]:
11: Edit libpath_jvm  :Lib path for libjvm [/usr/lib/jvm/java-8-openjdk-amd64/jre/lib/amd64 ↩
    /server]:
c: Cancel
w: Accept, write
Enter the choice [0-11, c, w]: w
Server gen ok!

$ make
javac -cp /usr/share/java/enduroxjava.jar:../ubftab/ubftab.jar Testsv.java
Testsv.java:20: error: cannot find symbol
        ub.Bchg(test.T_STRING_2_FLD, 0, "Hello World from XATMI server");
                ^
  symbol:   variable test
  location: class Testsv
1 error
Makefile:11: recipe for target 'Testsv.class' failed
make: *** [Testsv.class] Error 1
```

OK, we forgot to make symlink for the ubftab package from the runtime directory, thus we continue in same folder of **/opt/app-java/src/testsv**

```
$ ln -s /opt/app-java/ubftab /opt/app-java/src/ubftab

$ make 2>/dev/null
javac -cp /usr/share/java/enduroxjava.jar:../ubftab/ubftab.jar Testsv.java
jar -cmf manifest.txt testsv.jar ./Testsv.class
rm manifest.txt
exjld -n -o testsv -m 'Testsv' -L/usr/lib/jvm/java-8-openjdk-amd64/jre/lib/amd64 -L /usr/ ↩
    lib/jvm/java-8-openjdk-amd64/jre/lib/amd64/server testsv.jar
OS Version: linux wd: /opt/app-java/src/testsv/exjld.uUd78t
cc  -o /opt/app-java/src/testsv/testsv  jmain.c -lexjlds -ljava -ljvm -latmisrvinteg -latmi ↩
     -lubf -lnstd -lrt -ldl -lpthread -lm -lc -lpthread  -L/usr/lib/jvm/java-8-openjdk-amd64 ↩
    /jre/lib/amd64 -L/usr/lib/jvm/java-8-openjdk-amd64/jre/lib/amd64/server

$ ls -l
total 40
-rw-rw-r-- 1 user1 user1   871 Sep  8 13:33 Makefile
-rwxrwxr-x 1 user1 user1 13040 Sep  8 13:36 testsv
-rw-rw-r-- 1 user1 user1  1487 Sep  8 13:36 Testsv.class
-rw-rw-r-- 1 user1 user1  1296 Sep  8 13:36 testsv.jar
-rw-rw-r-- 1 user1 user1  1116 Sep  8 13:33 Testsv.java
-rw-rw-r-- 1 user1 user1  4885 Sep  8 13:33 ULOG.20190908
```

What we see in the end is that we have an binary **testsv** and the jar package **testsv.jar**.

The source code for basic Java XATMI server is quite simple, Check the **Testsv.java**:

```
import org.endurox.*;
import java.util.*;

public class Testsv implements Server, Service {
```

```
    public void tpService(AtmiCtx ctx, TpSvcInfo svcinfo) {

        ctx.tplogDebug("tpService/TESTSV called");
        TypedUbf ub = (TypedUbf)svcinfo.getData();

        //Print the buffer to stdout
        ub.tplogprintubf(AtmiConst.LOG_DEBUG, "Incoming request:");

        //Resize buffer, to have some more space
        ub.tprealloc(ub.Bused()+1024);

        //Add test field to buffer
        ub.Bchg(test.T_STRING_2_FLD, 0, "Hello World from XATMI server");

        //Reply OK back
        ctx.tpreturn(AtmiConst.TPSUCCESS, 0, svcinfo.getData(), 0);
    }

    public int tpSvrInit(AtmiCtx ctx, String [] argv) {
        ctx.tplogDebug("Into tpSvrInit()");

        ctx.tpadvertise("TESTSV", "Testsv", this);

        return AtmiConst.SUCCEED;
    }

    public void tpSvrDone(AtmiCtx ctx) {
        ctx.tplogDebug("Into tpSvrDone()");
    }

    public static void main(String[] args) {

        AtmiCtx ctx = new AtmiCtx();

        Testsv server = new Testsv();

        ctx.tprun(server);
    }
}
```

Basically we see that it has the standard java static main method which boot the object class which implements the **org.endurox.Server** and **org.endurox.Service** interfaces. All the driving is done by org.endurox.AtmiCtx class. The AtmiCtx.tprun() activates the the given object as an XATM server process. The server process receives calls of tpSvrInit() for init and tpSvrDone() for un-init. The init method advertises service "TESTSV".

The **Makefile** for the process looks like:

```
SOURCEDIR=.
SOURCES := $(shell find $(SOURCEDIR) -name '*.java')
CLASSES = $(addsuffix .class, $(basename $(SOURCES)))

BINARY=testsv
PKG=testsv.jar
MAINCLASS=Testsv

%.class: %.java
        javac -cp /usr/share/java/enduroxjava.jar:../ubftab/ubftab.jar $<

$(PKG): $(CLASSES)
        @echo "Manifest-Version: 1.0" > manifest.txt
        @echo "Main-Class: $(MAINCLASS)" >> manifest.txt
        @echo "" >> manifest.txt
```

```
        jar -cmf manifest.txt $(PKG) $(CLASSES)
        - rm manifest.txt
        exjld -n -o $(BINARY) -m '$(MAINCLASS)' -L/usr/lib/jvm/java-8-openjdk-amd64/jre/lib ↩
            /amd64 -L /usr/lib/jvm/java-8-openjdk-amd64/jre/lib/amd64/server $(PKG)

.PHONY: clean
clean:
        - rm *.class manifest.txt $(BINARY) $(PKG)

run:
        LD_LIBRARY_PATH=/usr/lib/jvm/java-8-openjdk-amd64/jre/lib/amd64:/usr/lib/jvm/java ↩
            -8-openjdk-amd64/jre/lib/amd64/server java -classpath /usr/share/java/ ↩
            enduroxjava.jar:./$(PKG) $(MAINCLASS)
```

But it would be possible to use the endurox.jar package from any other build tool like **ant** or **maven** with proper configuration.

5.3 Define Java client process

Client process in the same way as server process for demo purposes will be generated by Enduro/X generator.

```
$ cd /opt/app-java/src/testcl

$ xadmin gen java client 2>/dev/null
 0: classname    :Java class name of XATMI Client [Testcl]:
 1: pkgname      :Java package name [testcl.jar]:
 2: useubf       :Use UBF? [y]:
 3: svcname      :Service name to cal [TESTSV]:
 4: ubfname      :UBF package name [ubftab.jar]:
 5: config       :INI File section (optional, will read config if set) []:
 6: genmake      :Gen makefile [y]:
 7: enduroxjar   :Full path (incl filename) to enduroxjava.jar [/usr/share/java/enduroxjava ↩
     .jar]:
 8: makebin      :Link as binary [y]:
 9: binname      :Binary name [testcl]:
10: libpath_java :Lib path for libjava [/usr/lib/jvm/java-8-openjdk-amd64/jre/lib/amd64]:
11: libpath_jvm  :Lib path for libjvm [/usr/lib/jvm/java-8-openjdk-amd64/jre/lib/amd64/ ↩
    server]:

*** Review & edit configuration ***

 0: Edit classname    :Java class name of XATMI Client [Testcl]:
 1: Edit pkgname      :Java package name [testcl.jar]:
 2: Edit useubf       :Use UBF? [y]:
 3: Edit svcname      :Service name to cal [TESTSV]:
 4: Edit ubfname      :UBF package name [ubftab.jar]:
 5: Edit config       :INI File section (optional, will read config if set) []:
 6: Edit genmake      :Gen makefile [y]:
 7: Edit enduroxjar   :Full path (incl filename) to enduroxjava.jar [/usr/share/java/ ↩
     enduroxjava.jar]:
 8: Edit makebin      :Link as binary [y]:
 9: Edit binname      :Binary name [testcl]:
10: Edit libpath_java :Lib path for libjava [/usr/lib/jvm/java-8-openjdk-amd64/jre/lib/ ↩
    amd64]:
11: Edit libpath_jvm  :Lib path for libjvm [/usr/lib/jvm/java-8-openjdk-amd64/jre/lib/amd64 ↩
    /server]:
c: Cancel
w: Accept, write
Enter the choice [0-11, c, w]: w
Client gen ok!
```

```
$ make 2>/dev/null
javac -cp /usr/share/java/enduroxjava.jar:../ubftab/ubftab.jar Testcl.java
jar -cmf manifest.txt testcl.jar ./Testcl.class
rm manifest.txt
exjld -n -o testcl -m 'Testcl' -L/usr/lib/jvm/java-8-openjdk-amd64/jre/lib/amd64 -L /usr/ ↩
    lib/jvm/java-8-openjdk-amd64/jre/lib/amd64/server testcl.jar
OS Version: linux wd: /opt/app-java/src/testcl/exjld.mtJdC5
cc  -o /opt/app-java/src/testcl/testcl  jmain.c -lexjlds -ljava -ljvm -latmisrvinteg -latmi ↩
     -lubf -lnstd -lrt -ldl -lpthread -lm -lc -lpthread  -L/usr/lib/jvm/java-8-openjdk-amd64 ↩
    /jre/lib/amd64 -L/usr/lib/jvm/java-8-openjdk-amd64/jre/lib/amd64/server

$ ls -l
total 40
-rw-rw-r-- 1 user1 user1   871 Sep  8 13:46 Makefile
-rwxrwxr-x 1 user1 user1 13040 Sep  8 13:46 testcl
-rw-rw-r-- 1 user1 user1  1398 Sep  8 13:46 Testcl.class
-rw-rw-r-- 1 user1 user1  1281 Sep  8 13:46 testcl.jar
-rw-rw-r-- 1 user1 user1   975 Sep  8 13:46 Testcl.java
-rw-rw-r-- 1 user1 user1  4569 Sep  8 13:46 ULOG.20190908
```

For generator the defaults basically are used, but you may choose your own identifiers as well. The simple Java XATMI client process looks like this (**Testcl.java**):

```
import org.endurox.*;
import java.util.*;

public class Testcl {

    public void apprun(AtmiCtx ctx) {

        ctx.tplogDebug("apprun called");

        TypedUbf ub = (TypedUbf)ctx.tpalloc("UBF", "", 1024);

        //Add test field to buffer
        ub.Bchg(test.T_STRING_FLD, 0, "Hello World");

        try {
            ub = (TypedUbf)ctx.tpcall("TESTSV", ub, 0);
            //Print the buffer to stdout
            ub.tplogprintubf(AtmiConst.LOG_DEBUG, "Got response:");
        } catch (AtmiException e) {
            ctx.tplogInfo("got exception: %s", e.toString());
        }
    }

    public void appinit(AtmiCtx ctx) {
        ctx.tplogDebug("Into tpSvrInit()");

    }

    public void unInit(AtmiCtx ctx) {
        ctx.tpterm();
    }

    public static void main(String[] args) {

        AtmiCtx ctx = new AtmiCtx();

        Testcl cl = new Testcl();
        cl.appinit(ctx);
        cl.apprun(ctx);
```

```
            cl.unInit(ctx);
        }
}
```

Generate source is simple XATMI client process which prepares they key/value UBF buffer, adds some data and performs call to server. The Makefile for binary looks more or less in the same way as server Makefile.

Chapter 6

Running the example

To run the example, to bin directory needs to add sylinks for the compiled Java processes/binaries.

```
$ cd /opt/app-java/bin

$ ln -s /opt/app-java/src/testsv/testsv .

$ ln -s /opt/app-java/src/testcl/testcl .
```

After this we are ready to boot up the application, thus lets load the environment and start up.

```
$ cd /opt/app-java/conf

$ . setappj

$ xadmin start -y
Enduro/X 7.0.3, build Sep  1 2019 18:26:27, using epoll for LINUX (64 bits)

Enduro/X Middleware Platform for Distributed Transaction Processing
Copyright (C) 2009-2016 ATR Baltic Ltd.
Copyright (C) 2017-2019 Mavimax Ltd. All Rights Reserved.

This software is released under one of the following licenses:
AGPLv3 or Mavimax license for commercial use.

* Shared resources opened...
* Enduro/X back-end (ndrxd) is not running
* ndrxd PID (from PID file): 12993
* ndrxd idle instance started.
exec testsv -k 0myWI5nu -i 20 -e /opt/app-java/log/testsv.log -r --  :
        process id=12995 ... Started.
exec testsv -k 0myWI5nu -i 120 -e /opt/app-java/log/java.log -r --  :
        process id=13009 ... Started.
Startup finished. 2 processes started.
```

As it could be seen both Java server processes are booted. Let's check their services:

```
$ xadmin psc

Nd Service Name Routine Name Prog Name SRVID #SUCC #FAIL MAX      LAST     STAT
-- ------------ ------------ --------- ----- ----- ----- -------- -------- -----
1  TESTSV       Testsv       testsv    20    0     0     0ms      0ms      AVAIL
1  TESTSV       Testsv       testsv    120   0     0     0ms      0ms      AVAIL
```

Both servers provide **TESTSV** service. Now lets perform the test by calling the Java client process in classical way:

```
$ cd /opt/app-java/src/testcl

$ java -classpath /usr/share/java/enduroxjava.jar:./testcl.jar Testcl
t:USER:5:d5d3db3a:13664:7f46d18b1700:000:20190908:141009742:tplog       :/tplog.c:0536:Into ↩
     tpSvrInit()
t:USER:5:d5d3db3a:13664:7f46d18b1700:000:20190908:141009742:tplog       :/tplog.c:0536: ↩
    apprun called
t:USER:5:d5d3db3a:13664:7f46d18b1700:001:20190908:141009743:plogprintubf:_tplog.c:0100:Got  ↩
    response:
T_STRING_FLD    Hello World
T_STRING_2_FLD  Hello World from XATMI server
```

As ubf tables are generated as **final** constants, their values are copied to destination byte code (.class). Thus there is no need for reference to them.

Now call linked Java executable:

```
$ cd /opt/app-java/bin
$ ./testcl
t:USER:5:d5d3db3a:13709:7f6ebd12d800:000:20190908:141122288:tplog       :/tplog.c:0536:Into ↩
     tpSvrInit()
t:USER:5:d5d3db3a:13709:7f6ebd12d800:000:20190908:141122289:tplog       :/tplog.c:0536: ↩
    apprun called
t:USER:5:d5d3db3a:13709:7f6ebd12d800:001:20190908:141122289:plogprintubf:_tplog.c:0100:Got  ↩
    response:
T_STRING_FLD    Hello World
T_STRING_2_FLD  Hello World from XATMI server
```

From both run attempts we see the same results. Thus it is developer preference to use linked binaries for simple server delivery, or use classical way with java runner.

Chapter 7

Conclusions

In the end we see that it is quite simple to create XATMI Java client and server processes. The good thing is that this API is consistent with Go and C/C++ languages. Thus any of these three programming languages can be mixed in single high performance application.

This tutorial shows only basic features of the Enduro/X. There is more to study as async calls, **tpforward(3)**, persistent queues, events, distributed transactions and more. For full API consult the Enduro/X API Java doc pages. Also unit tests can give a clue for the full functionality use.