Getting Started Tutorial

REVISION HISTORY			
NUMBER	DATE	DESCRIPTION	NAME
1.1	2018-11	Updates for release 6.0 - move to provision script	MV

Contents

Chapter 1

About the guide

Getting started tutorial covers Enduro/X installation from binary package, environment setup, creating a sample Enduro/X server and creating sample client. This include creating a necessary configuration files. Finally the application is booted, and client process calls the sample server 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 package (RPM for Centos 6.x)

We will also need a gcc compiler to build our bankapp

```
# wget https://www.endurox.org/attachments/download/243/endurox-5.3.6-1.centos6_9_GNU. ↩
    x86_64.rpm
# rpm -i endurox-5.3.6-1.centos6_9_GNU.x86_64.rpm
# yum install gcc
```

Chapter 4

Configuring the application environment

We will run our app "app1" from new user "user1". Application domain will be located in /opt/app1 folder.

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

```
# useradd -m user1
# mkdir -p /opt/app1
# chown user1 /opt/app1
# su - user1
$ cd /opt/app1
$ xadmin provision -d \
    -v qprefix=app1 \
    -v installQ=n  \
    -v eventSv=n \
    -v cpmSv=n \
    -v configSv=n \
    -v bridge=n \
    -v addubf=bank.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	app1	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	bank.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/app1/test
$ mkdir -p /opt/app1/src/bankcl
$ mkdir -p /opt/app1/src/banksv
```

Thus the final directory structure built for the application is

- /opt/app1/conf - will contain configuration files.
- /opt/app1/src/bankcl - Enduro/X sample client process source
- /opt/app1/src/banksv - Enduro/X sample server process sources.
- /opt/app1/bin - for executables.
- /opt/app1/ubftab - for tables for field definitions.
- /opt/app1/tmp - temp dir
- /opt/app1/log - for logging
- /opt/app1/test - test data

For demo purposes the provision script have made more or less empty XATMI server configuration file found in **/opt/app1/conf/ndrxcon**
Lets register firstly our XATMI server named **banksv** here first. Do this in **<servers/>** section add following **<server />** block in the file:

```
<?xml version="1.0" ?>
<endurox>
...
        </defaults>
        <servers>
                <server name="banksv">
                        <srvid>1</srvid>
                        <min>2</min>
                        <max>2</max>
                        <sysopt>-e /opt/app1/log/BANKSV -r</sysopt>
                </server>
        </servers>
</endurox>
```

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/app1/conf/app.ini):

```
[@debug]
..
banksv= ndrx=5 ubf=0 tp=5 file=${NDRX_APPHOME}/log/BANKSV
bankcl= ndrx=5 ubf=0 tp=5 file=${NDRX_APPHOME}/log/BANKCL
ud= ndrx=5 ubf=0 file=${NDRX_APPHOME}/log/UD
```

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).

If at this step we try to boot an application server, it should start fine, except that **banksv** binary won't be found:

```
$ cd /opt/app1/conf
$ source setapp1
$ xadmin start -y
Enduro/X 5.4.1, build Nov  7 2018 08:48:27, using SystemV for LINUX (64 bits)

Enduro/X Middleware Platform for Distributed Transaction Processing
Copyright (C) 2009-2016 ATR Baltic Ltd.
Copyright (C) 2017,2018 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): 4716
* ndrxd idle instance started.
exec banksv -k 0myWI5nu -i 1 -e /opt/app1/log/BANKSV -r --  :
        process id=4720 ... Died.
exec banksv -k 0myWI5nu -i 2 -e /opt/app1/log/BANKSV -r --  :
        process id=4723 ... Died.
Startup finished. 0 processes started.
```

This is ok, we have configured two copies of **banksv** Enduro/X servers, which we are not yet built, thus we get the error.

If you run 'xadmin' and get following error:

```
$ xadmin
Failed to initialize!
```

Then this typically means, that you do not have configure operating system properly see **Operating System configuration** section. More info is logged to /opt/app1/log/xadmin.log

Chapter 5

Creating the server process

Firstly to create a "bank" server, we will have to define the fields in which we will transfer the data. We will need following fields:

- T_ACCNUM - Account number, type string
- T_ACCCUR - Account currency, type string
- T_AMTAVL - Available balance in account, type double

So we will create a service "BALANCE" to which we will T_ACCNUM and T_ACCCUR. The process will return balance in T_AMTAVL.

5.1 Defining the UBF fields

Required fields will be define into /opt/app1/ubftab/bank.fd with following contents:

```
$/* ------------------------------------------------------------------------
$** Bank app field definitions for UBF buffer
$** ------------------------------------------------------------------------
$*/

$#ifndef __BANK_H
$#define __BANK_H

*base 1000

#NAME           ID      TYPE    FLAG    COMMENT
#----           --      ----    ----    -------
# Service name for UD
T_ACCNUM        1       string  -       Account number
T_ACCCUR        2       string  -       Account currency
T_AMTAVL        3       double  -       Account balance

$#endif
```

To generate C header fields for UBF buffer, run ‘mkfldhdr’ command in /opt/app1/ubftab folder:

```
$ mkfldhdr
...
NDRX:5: 2038:000:20151116:033733008:fldhdr.c:0290:Finished with : SUCCESS
$ ls -l
total 16
```

```
-rw-r--r--. 1 user1 user1  459 Nov 16 03:36 bank.fd
-rw-rw-r--. 1 user1 user1  525 Nov 16 03:37 bank.fd.h
-rw-r--r--. 1 user1 user1 3704 Nov 16 03:18 Exfields
-rw-rw-r--. 1 user1 user1 3498 Nov 16 03:37 Exfields.h
```

5.2 Server source code

We will have sample server process which will just print in trace file account, currency. In return it will set "random" balance in field "T_AMTAVL". The source code of /opt/app1/src/banksv/banksv.c looks as follows:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>

/* Enduro/X includes: */
#include <atmi.h>
#include <ubf.h>
#include <bank.fd.h>

#define SUCCEED         0
#define FAIL            -1

/**
 * BALANCE service
 */
void BALANCE (TPSVCINFO *p_svc)
{
        int ret=SUCCEED;
        double balance;
        char account[28+1];
        char currency[3+1];
        BFLDLEN len;

        UBFH *p_ub = (UBFH *)p_svc->data;

        fprintf(stderr, "BALANCE got call\n");

        /* Resize the buffer to have some space in... */
        if (NULL==(p_ub = (UBFH *)tprealloc ((char *)p_ub, 1024)))
        {
                fprintf(stderr, "Failed to realloc the UBF buffer - %s\n",
                        tpstrerror(tperrno));
                ret=FAIL;
                goto out;
        }


        /* Read the account field */
        len = sizeof(account);
        if (SUCCEED!=Bget(p_ub, T_ACCNUM, 0, account, &len))
        {
                fprintf(stderr, "Failed to get T_ACCNUM[0]! -  %s\n",
                        Bstrerror(Berror));
                ret=FAIL;
                goto out;
        }

        /* Read the currency field */
```

```
        len = sizeof(currency);
        if (SUCCEED!=Bget(p_ub, T_ACCCUR, 0, currency, &len))
        {
                fprintf(stderr, "Failed to get T_ACCCUR[0]! -  %s\n",
                        Bstrerror(Berror));
                ret=FAIL;
                goto out;
        }

        fprintf(stderr, "Got request for account: [%s] currency [%s]\n",
                        account, currency);

        srand(time(NULL));
        balance = (double)rand()/(double)RAND_MAX + rand();

        /* Return the value in T_AMTAVL field */

        fprintf(stderr, "Retruning balance %lf\n", balance);


        if (SUCCEED!=Bchg(p_ub, T_AMTAVL, 0, (char *)&balance, 0L))
        {
                fprintf(stderr, "Failed to set T_AMTAVL! -  %s\n",
                        Bstrerror(Berror));
                ret=FAIL;
                goto out;
        }

out:
        tpreturn(  ret==SUCCEED?TPSUCCESS:TPFAIL,
                0L,
                (char *)p_ub,
                0L,
                0L);
}

/**
 * Do initialization
 */
int tpsvrinit(int argc, char **argv)
{
        if (SUCCEED!=tpadvertise("BALANCE", BALANCE))
        {
                fprintf(stderr, "Failed to advertise BALANCE - %s\n",
                        tpstrerror(tperrno));
                return FAIL;
        }

        return SUCCEED;
}

/**
 * Do de-initialization
 */
void tpsvrdone(void)
{
        fprintf(stderr, "tpsvrdone called\n");
}
```

Very simple Makefile will look like (/opt/app1/src/banksv/Makefile):

```
banksv: banksv.c
```

```
        cc -o banksv banksv.c -I. -I ../../ubftab -latmisrv -latmi -lubf -lnstd -lrt -ldl - ↩
            lm
```

Build the binary:

```
$ cd /opt/app1/src/banksv
$ make
cc -o banksv banksv.c -I. -I ../../ubftab -latmisrv -latmi -lubf -lnstd -lrt -ldl -lm
ls -l
total 20
-rwxrwxr-x. 1 user1 user1 9937 Nov 16 04:11 banksv
-rw-rw-r--. 1 user1 user1 1926 Nov 16 04:07 banksv.c
-rw-rw-r--. 1 user1 user1  105 Nov 16 04:01 Makefile
```

So binary is built next we will try to start it.

5.3 Booting the server process

To start the binary, first we need to copy it to binary directory:

```
$ cp /opt/app1/src/banksv/banksv /opt/app1/bin/banksv
```

Now start it with "xadmin start". This will cause to boot any unbooted processes to start (which previosly did not start because we didn't have 'banksv' executable in bin directory.

```
$ xadmin tart -y
Enduro/X 5.4.1, build Nov 12 2018 08:11:50, using SystemV for LINUX (64 bits)

Enduro/X Middleware Platform for Distributed Transaction Processing
Copyright (C) 2009-2016 ATR Baltic Ltd.
Copyright (C) 2017,2018 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): 22128
* ndrxd idle instance started.
exec banksv -k 0myWI5nu -i 1 -e /opt/app1/log/BANKSV -r --  :
        process id=22132 ... Started.
exec banksv -k 0myWI5nu -i 2 -e /opt/app1/log/BANKSV -r --  :
        process id=22136 ... Started.
Startup finished. 2 processes started.
```

To check that our BALANCE service is advertised, we can execute command "xadmin psc" - print services:

```
$ xadmin psc
Enduro/X 5.4.1, build Nov 12 2018 08:11:50, using SystemV for LINUX (64 bits)

Enduro/X Middleware Platform for Distributed Transaction Processing
Copyright (C) 2009-2016 ATR Baltic Ltd.
Copyright (C) 2017,2018 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...
* ndrxd PID (from PID file): 22128
Nd Service Name Routine Name Prog Name SRVID #SUCC #FAIL MAX      LAST     STAT
```

```
-- ------------ ------------ --------- ----- ----- ----- -------- -------- -----
1  BALANCE      BALANCE      banksv    1     0     0     0ms      0ms      AVAIL
1  BALANCE      BALANCE      banksv    2     0     0     0ms      0ms      AVAIL
```

We see here two copies for **banksv** binaries running (Server ID 1 & 2). Both advertises "BALANCE" service.

5.4 Testing the service with "ud" command

It is possible to call the service with out a client process. This is useful for testing. Service can be called with ‘ud’ utility. In which we define the target service name and any additional UBF buffer fields. In our case these fields are T_ACCNUM and T_ACCCUR, which are mandatory for the service. So we will create a ‘test.ud’ file in folder /opt/app1/test. /opt/app1/test/test.ud looks like:

```
SRVCNM  BALANCE
T_ACCNUM        ABC123467890
T_ACCCUR        EUR
```

To call the service just pipe the data to the ‘ud’:

```
$ ud < /opt/app1/test/test.ud
SENT pkt(1) is :
T_ACCNUM        ABC123467890
T_ACCCUR        EUR

RTN pkt(1) is :
T_AMTAVL        1355808545.118969
T_ACCNUM        ABC123467890
T_ACCCUR        EUR
```

We see that our "dummy" balance returned is "1355808545.118969". So test service is working ok. Now we should write a client app, which could call the service via tpcall() XATMI API call.

Chapter 6

Creating the client application

Bank client application will setup T_ACCNUM and T_ACCCUR fields and will call "BALANCE" service, after the call client application will print the balance on screen.

6.1 Client binary source code

Code for client application: /opt/app1/src/bankcl/bankcl.c

```
#include <string.h>
#include <stdio.h>
#include <stdlib.h>
#include <memory.h>
#include <math.h>

#include <atmi.h>
#include <ubf.h>
#include <bank.fd.h>

#define SUCCEED         0
#define FAIL            -1

/**
 * Do the test call to the server
 */
int main(int argc, char** argv) {

        int ret=SUCCEED;
        UBFH *p_ub;
        long rsplen;
        double balance;

        /* allocate the call buffer */
        if (NULL== (p_ub = (UBFH *)tpalloc("UBF", NULL, 1024)))
        {
                fprintf(stderr, "Failed to realloc the UBF buffer - %s\n",
                        tpstrerror(tperrno));
                ret=FAIL;
                goto out;
        }

        /* Set the data */
        if (SUCCEED!=Badd(p_ub, T_ACCNUM, "ACC00000000001", 0) ||
                SUCCEED!=Badd(p_ub, T_ACCCUR, "USD", 0))
```

```
        {
                fprintf(stderr, "Failed to get T_ACCNUM[0]! -  %s\n",
                        Bstrerror(Berror));
                ret=FAIL;
                goto out;
        }

        /* Call the server */
        if (FAIL == tpcall("BALANCE", (char *)p_ub, 0L, (char **)&p_ub, &rsplen,0))
        {
                fprintf(stderr, "Failed to call BALANCE - %s\n",
                        tpstrerror(tperrno));

                ret=FAIL;
                goto out;
        }

        /* Read the balance field */

        if (SUCCEED!=Bget(p_ub, T_AMTAVL, 0, (char *)&balance, 0L))
        {
                fprintf(stderr, "Failed to get T_AMTAVL[0]! -  %s\n",
                        Bstrerror(Berror));
                ret=FAIL;
                goto out;
        }

        printf("Account balance is: %.2lf USD\n", balance);

out:
        /* free the buffer */
        if (NULL!=p_ub)
        {
                tpfree((char *)p_ub);
        }

        /* Terminate ATMI session */
        tpterm();
        return ret;
}
```

Makefile (/opt/app1/src/bankcl/Makefile) looks like:

```
bankcl: bankcl.c
        cc -o bankcl bankcl.c -I. -I ../../ubftab -latmiclt -latmi -lubf -lnstd -lrt -ldl - ↩
            lm
```

Once both bankcl.c and Makefile is created, you can run the build process:

```
$ cd /opt/app1/src/bankcl
$ make
$ ls -l
total 20
-rwxrwxr-x. 1 user1 user1 9471 Nov 22 13:34 bankcl
-rw-rw-r--. 1 user1 user1 1380 Nov 22 13:34 bankcl.c
-rw-rw-r--. 1 user1 user1  105 Nov 22 13:32 Makefile
```

6.2 Running the client process

We will start the application from the same build directory. The results are following:

```
$ /opt/app1/src/bankcl/bankcl
Account balance is: 883078058.68 USD
```

Chapter 7

Deploy "banksv" in Docker container

To deploy the example files in Docker container, follow the instructions found in endurox-docker repository: https://github.com/-endurox-dev/endurox-docker

Chapter 8

Conclusions

From the above sample it could be seen that creating a ATMI application is pretty easy and straight forward. This application was very basic, just doing the call to Enduro/X service. However the same application could work in cluster, where "BALANCE" service can be located on different physical machine and 'bankcl' will still work, as platform will ensure the visibility of the "BALANCE" service, see the [TPBRIDGE] for clustering. The source files of the sample app are located in "getting_started_tutorial-files/opt/app1" folder.

Chapter 9

Additional documentation

9.1 Internet resources

[1] [ATMI-API] http://docs.oracle.com/cd/E13203_01/tuxedo/tux71/html/pgint6.htm

[2] [FML-API] http://docs.oracle.com/cd/E13203_01/tuxedo/tux91/fml/index.htm

[3] [EX_ADMINGUIDE] ex_adminman(guides)

[4] [MQ_OVERVIEW] man 7 mq_overview

[5] [EX_ENV] ex_env(5)

[6] [NDRXCONFIG] ndrxconfig.xml(5)

[7] [DEBUGCONF] ndrxdebug.conf(5)

[8] [XADMIN] xadmin(8)

[9] [TPBRIDGE] tpbridge(8)

Chapter 10

Glossary

This section lists

ATMI
: Application Transaction Monitor Interface

UBF
: Unified Buffer Format it is similar API as Tuxedo's FML