Getting Started Tutorial

REVISION HISTORY			
NUMBER	DATE	DESCRIPTION	NAME
1.0	2015-11	Initial draft	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 neccessary configuration files. Finally the applicatioin is booted, and client process calls the sample server process.

Chapter 2

Configuring Linux Kernel parameters

Kernel parameter configuration is needed for Enduro/X runtime. It includes changing the security and Posix queue settings.

Security Configuration

```
$ su - root
# cat << EOF >> /etc/security/limits.conf

# Do not limit message Q Count.
# Some Linux 3.x series kernels have a bug, that limits 1024
# queues for one system user.
# In 2.6.x and 4.x this is fixed, to have
# unlimited count of queues (memory limit).
# ealrier and later Linuxes have fixed this issue.
*               soft    msgqueue        -1
*               hard    msgqueue        -1

# Increase the number of open files
*               soft    nofile  1024
*               hard    nofile  65536

EOF
```

Message Queue Configuration

At the startup of the system needs to mount a Posix Queues folder, and needs to set a appropriate limits. To do this automatically at system startup, Linuxes which supports */etc/rc.local*, must add following lines before "exit 0":

```
# Mount the /dev/mqueue
mkdir /dev/mqueue
mount -t mqueue none /dev/mqueue

# Max Messages in Queue
echo 10000 > /proc/sys/fs/mqueue/msg_max

# Max message size (Currently Enduro/X supports only 32K as max)
echo 32000 > /proc/sys/fs/mqueue/msgsize_max

# Max number of queues for user
echo 10000 > /proc/sys/fs/mqueue/queues_max
```

Chapter 3

Installing binary package (RPM for Centos 6.x)

We will also need a gcc compiler to build our bankapp

```
# wget http://www.endurox.org/attachments/download/6/endurox-2.3.2-1.centos6.x86_64.rpm
# rpm -i endurox-2.3.2-1.centos6.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. With follwing direcotry structure:

- /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 defintions.
- /opt/app1/tmp - temp dir
- /opt/app1/log - for logging
- /opt/app1/test - test data

Create the user & directories:

```
# useradd -m user1
# mkdir -p /opt/app1/conf
# mkdir -p /opt/app1/src/bankcl
# mkdir -p /opt/app1/src/banksv
# mkdir -p /opt/app1/bin
# mkdir -p /opt/app1/ubftab
# mkdir -p /opt/app1/tmp
# mkdir -p /opt/app1/log
# mkdir -p /opt/app1/test
# chown -R user1 /opt/app1
```

Next we will configure environment files, file /opt/app1/conf/setndrx - this will contain neccessary configuration for Enduro/X.

Copy following text to /opt/app1/conf/setndrx

```
#!/bin/bash
export NDRX_NODEID=1
# If 1 - then yes, if 0 - then not clusterised.
export NDRX_CLUSTERISED=0
# Load balance, 0 = process all locally, 100 = process all on remote servers
export NDRX_LDBAL=0
# tpcall() timeout:
export NDRX_TOUT=60
# where to write ulog
export NDRX_ULOG=/opt/app1/log
export NDRX_QPREFIX=/app1
export NDRX_SVCMAX=20000
```

```
export NDRX_SRVMAX=10000
export NDRX_QPATH=/dev/mqueue
export NDRX_SHMPATH=/dev/shm
# Milli seconds to wait for command
export NDRX_CMDWAIT=1
export NDRX_DPID=/opt/app1/tmp/ndrxd.pid
# Random key to indentify the processes beloning to this session (i.e. used in ps ef)
export NDRX_RNDK="0myWI5nu"
# System V Semaphores...
export NDRX_IPCKEY=44000
# Posix queue config (attribs..)
# Max number of messages that can be put in one queue
export NDRX_MSGMAX=1000
# Daemon Q size...
export NDRX_DQMAX=100
# Max message size (in bytes), max about ~10 MB on Linux
# Stack size must be at least NDRX_MSGSIZEMAX*30
export NDRX_MSGSIZEMAX=10000
# Where app domain lives
export NDRX_APPHOME=/opt/app1
# Where NDRX runtime lives
export NDRX_HOME=/usr
# Debug config too
export NDRX_DEBUG_CONF=/opt/app1/conf/debug.conf
# NDRX config too.
export NDRX_CONFIG=/opt/app1/conf/ndrxconfig.xml
export PATH=$PATH:/opt/app1/bin
export FLDTBLDIR=/opt/app1/ubftab
# Max fields for hashing UBF
export NDRX_UBFMAXFLDS=16000

# Log & levels (basic for scripting..)
export NDRX_DMNLOG=/opt/app1/log/ndrxd.log
export NDRX_DMNLEV=5

export NDRX_LOG=/opt/app1/log/xadmin.log
export NDRX_LEV=5

# Correct the path so that ndrx can find ndrxd
export PATH=$PATH:$NDRX_HOME/bin

# UBFTAB Exfields - Enduro/X specifc, bank.fd - our apps' UBF fields
export FIELDTBLS=Exfields,bank.fd
```

Basic application server configuration (/opt/app1/conf/ndrxconfig.xml)

```
<?xml version="1.0" ?>
<endurox>
    <appconfig>
         <!-- ALL BELLOW ONES USES <sanity> periodical timer  -->
         <!-- Sanity check time, sec -->
         <sanity>5</sanity>
         <!--
             Seconds in which we should send service refresh to other node.
         -->
         <brrefresh>6</brrefresh>

         <!--  <sanity> timer, end -->

         <!-- ALL BELLOW ONES USES <respawn> periodical timer  -->
         <!-- Do dead process restart every X seconds
         NOT USED ANYMORE, REPLACED WITH SANITY!
```

```
        <respawncheck>10</respawncheck>
        -->
        <!-- Do process reset after 1 sec -->
        <restart_min>1</restart_min>
        <!-- If restart fails, then boot after +5 sec of previous wait time -->
        <restart_step>1</restart_step>
        <!-- If still not started, then max boot time is a 30 sec. -->
        <restart_max>5</restart_max>
        <!--  <sanity> timer, end -->

        <!-- Time after attach when program will start do sanity & respawn checks,
             starts counting after configuration load -->
        <restart_to_check>20</restart_to_check>

        <!-- Setting for pq command, should ndrxd collect service
             queue stats automatically
        If set to Y or y, then queue stats are on.
        Default is off.
        -->
        <gather_pq_stats>Y</gather_pq_stats>

    </appconfig>
<defaults>
    <min>1</min>
    <max>2</max>
    <!-- Kill the process which have not started in <start_max> time -->
    <autokill>1</autokill>
    <!--
    <respawn>1<respawn>
    -->
    <!--
        <env></env> works here too!
    -->
     <!-- The maximum time while process can hang in 'starting' state i.e.
        have not completed initialization, sec
        X <= 0 = disabled
    -->
     <start_max>2</start_max>
     <!--
        Ping server in every X seconds (step is <sanity>).
     -->
     <pingtime>1</pingtime>
     <!--
        Max time in seconds in which server must respond.
        The granularity is sanity time.
        X <= 0 = disabled
     -->
     <ping_max>4</ping_max>
     <!--
        Max time to wait until process should exit on shutdown
        X <= 0 = disabled
     -->
     <end_max>5</end_max>
     <!-- Interval, in seconds, by which signal sequence -2, -15, -9, -9.... will be ↩
         sent
     to process until it have been terminated. -->
     <killtime>1</killtime>
     <!-- List of services (comma separated) for ndrxd to export services over bridges ↩
         -->
<!--     <exportsvcs>FOREX</exportsvcs> -->
    </defaults>
    <servers>
```

```
                <!-- This is binary we are about to build -->
                <server name="banksv">
                        <srvid>1</srvid>
                        <min>2</min>
                        <max>2</max>
                        <sysopt>-e /opt/app1/log/BANKSV -r</sysopt>
                </server>
        </servers>
</endurox>
```

Setup debug config (/opt/app1/conf/debug.conf):

```
* ndrx=1 ubf=0 lines=1 bufsz=1000 file=
xadmin file=${NDRX_APPHOME}/log/xadmin.log
ndrxd file=${NDRX_APPHOME}/log/ndrxd.log
banksv file=${NDRX_APPHOME}/log/BANKSV
bankcl file=${NDRX_APPHOME}/log/BANKCL
```

We will get the default Exfields version and will try to start the app server, should start, except 'banksv' will not be found:

```
$ cp /usr/share/endurox/ubftab/Exfields /opt/app1/ubftab
$ cd /opt/app1/conf
$ chmod +x setndrx
$ . setndrx
$ xadmin start -y

Enduro/X v2.3.2, build Nov 16 2015 08:22:23

Enduro/X Middleware Platform for Distributed Transaction Processing
Copyright (C) 2015, Mavimax, Ltd. All Rights Reserved.

This software is released under one of the following licenses:
GPLv2 (or later) or Mavimax's license for commercial use.

EnduroX back-end (ndrxd) is not running!
ndrxd PID (from PID file): 1695
ndrxd idle instance started!
exec banksv -k 0myWI5nu -i 1 -e /opt/app1/log/BANKSV -r --  :
        process id=1696 ... No such file or directory.
exec banksv -k 0myWI5nu -i 2 -e /opt/app1/log/BANKSV -r --  :
        process id=1698 ... No such file or directory.
Startup finished. 0 processes started.
```

This is ok, we have configured two copies of eserver 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 run /etc/rc.local (either by root or by reboot). 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.

Defining the UBF fields

Requried fields will be define into /opt/app1/ubftab/bank.fd with follwing contents:

```
$/* ------------------------------------------------------------------------
$** Bank app field defintions 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:0265:Output directory is [.]
NDRX:5: 2038:000:20151116:033733008:fldhdr.c:0277:Use environment variables
NDRX:5: 2038:000:20151116:033733008:fldhdr.c:0337:enter generate_files()
NDRX:5: 2038:000:20151116:033733008:fldhdr.c:0395:/opt/app1/ubftab/Exfields processed OK,  ↩
    output: ./Exfields.h
NDRX:5: 2038:000:20151116:033733008:fldhdr.c:0395:/opt/app1/ubftab/bank.fd processed OK,  ↩
    output: ./bank.fd.h
```

```
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
```

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

Build the binary:

```
$ cd /opt/app1/src/banksv
$ make
gcc -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.

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 start
Enduro/X v2.3.2, build Nov 16 2015 08:22:23

Enduro/X Middleware Platform for Distributed Transaction Processing
Copyright (C) 2015, Mavimax, Ltd. All Rights Reserved.

This software is released under one of the following licenses:
GPLv2 (or later) or Mavimax's license for commercial use.

ndrxd PID (from PID file): 2608
Are you sure you want to start application? [Y/N]: y
exec banksv -k 0myWI5nu -i 1 -e /opt/app1/log/BANKSV -r --  :
        process id=2617 ... Started.
exec banksv -k 0myWI5nu -i 2 -e /opt/app1/log/BANKSV -r --  :
        process id=2618 ... Started.
Startup finished. 2 processes started.
```

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

```
$ xadmin psc
Enduro/X v2.3.2, build Nov 16 2015 08:22:23

Enduro/X Middleware Platform for Distributed Transaction Processing
Copyright (C) 2015, Mavimax, Ltd. All Rights Reserved.

This software is released under one of the following licenses:
GPLv2 (or later) or Mavimax's license for commercial use.

ndrxd PID (from PID file): 2608
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 advertizes "BALANCE" service.

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.

Client binary source code

Code for client application: /opt/app1/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
        gcc -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
```

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

Conclusions

From the above sample it could be seen that creating a ATMI application ir pretty easy and stright 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 8

Additional documentation

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_OVERVIEW] ex_overview.pdf

[4] [MQ_OVERVIEW] man 7 mq_overview

[5] [EX_ENV] man 5 ex_env or ex_env.pdf

[6] [NDRXCONFIG] man 5 ndrxconfig.xml or ndrxconfig.xml.pdf

[7] [DEBUGCONF] man 5 ndrxdebug.conf or ndrxdebug.conf.pdf

[8] [XADMIN] man 8 xadmin or xadmin.pdf

[9] [TPBRIDGE] man 8 tpbridge or tpbridge.pdf

Chapter 9

Glossary

This section lists

ATMI
: Application Transaction Monitor Interface

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