Enduro/X Java Internal Developer Guide

REVISION HISTORY

NUMBER	DATE	DESCRIPTION	NAME
1.0	2018-08	Initial draft	MV

Contents

Chapter 1

Intro

This document is for external and internal purposes of the Enduro/X Java module developer. It can be used for building the Enduro/X Java package for further use. Document also contains the notes for Enduro/X Java module developer.

It contains solutions for main pitfalls found during the development process. Also document contains main configuration steps to get the development IDE working.

Document starts with approach of building the Enduor/X Java module for external use. Afterwards document describes the process for preparing the IDE for internal module development process.

Chapter 2

Installing Java JDK

First of all to start using or developing Enduro/X Java language plugin, the JDK needs to be installed. Enduro/X supports JDK version 1.7 and above.

2.1 CentOS/RHEL/Oracle Linux 7.X

To install the JDK for RHEL system, use following command:

```
# yum install java-1.8.0-openjdk java-devel
```

2.2 CentOS/RHEL/Oracle Linux 8.X

To install the JDK for RHEL system, use following command:

```
# yum install java-11-openjdk-devel
```

2.3 Ubuntu systems

```
$ sudo apt-get install default-jdk
```

2.4 SLES

Example version:

```
$ sudo zypper install java-10-openjdk java-10-openjdk-devel
```

2.5 FreeBSD systems

```
$ sudo pkg install openjdk8
```

Chapter 3

Prepare Enduro/X for Build

The preparation for build also includes the step of downloading PostgreSQL JDBC drivers, as these are used (if not disabled) for testing purposes of the XA transactions. PostgreSQL database shall be already configured and users prepared. The preparation process is described in building_guide(guides)(Building Enduro/X On GNU/Linux Platform, Testing PostgreSQL chapter). Also note that environment variables in ~/ndrx_home must be configured i.e. EX_PG_HOST, etc. variables must be configured in order the to establish the connection to database.

3.1 Getting the source

```
$ git clone https://github.com/endurox-dev/endurox-java
$ cd endurox-java
```

3.2 Getting the PostgreSQL JDBC driver

Download the corresponding version from https://jdbc.postgresql.org/download.html. For example version postgresql-42.2.6.jar.

By assuming that you are in the endurox-java folder:

```
$ cd tests/03_xapostgres/jdbcdrivers
$ wget https://jdbc.postgresql.org/download/postgresql-42.2.6.jar
$ mv postgresql-42.2.6.jar pgjdbc.jar
```

3.3 Getting the Oracle JDBC Driver

If using having Oracle DB installed and ready for testing, then JDBC driver must be installed too. This can be downloaded from https://www.oracle.com/database/technologies/appdev/jdbc-downloads.html

In the end, what Enduro/x needs is "ojdbc.jar" to be located in the project path:

```
$ ls -l endurox-java/tests/02_xaoracle/jdbcdrivers/ojdbc.jar
```

3.4 Update environment variables

Enduro/X Java module requires libjava.so and libjvm.so (and other OS counterparts), Thus the runtime library search path must be set correspondingly. To execute the Enduro/X tests, the ~/ndrx_home source script must be updated:

For example on Oracle Linux 7:

```
$ vi ~/ndrx_home

Add:

# Java settings
export JAVA_HOME=/usr/lib/jvm/java-1.8.0-openjdk-1.8.0.222.b10-0.el7_6.x86_64

# Add or update:
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:$JAVA_HOME/jre/lib/amd64
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:$JAVA_HOME/jre/lib/amd64/server
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/home/user1/modules/endurox-java/libsrc/c:/home/ ↩
    user1/modules/endurox-java/libexjlds
```

For Mac OS:

```
$ vi ~/ndrx_home

Add:

# Java settings
export JAVA_HOME=/Library/Java/JavaVirtualMachines/jdk1.8.0_221.jdk/Contents/Home


# Add or update:
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
export DYLD_LIBRARY_PATH=$DYLD_LIBRARY_PATH:/Users/user1/modules/endurox-java/libsrc/c:/ ↩
    Users/user1/modules/endurox-java/libexjlds
```

NOTE: that "..A/Resources:" must be before java library path, otherwise expect such errors as "Symbol not found __cg_jpeg_resync_to_

Oracle Linux 8 with Java 11:

```
$ vi ~/ndrx_home

Add:


# Java settings
export JAVA_HOME=/usr/lib/jvm/java-11-openjdk-11.0.4.11-0.el8_0.x86_64

# Add or update:
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:$JAVA_HOME/lib
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:$JAVA_HOME/lib/server
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/home/user1/modules/endurox-java/libsrc/c:/home/ ↩
    user1/modules/endurox-java/libexjlds
```

Suse Enterprise Linux Server 15 (SLES):

```
$ vi ~/ndrx_home

Add:
```

```
# Java settings
export JAVA_HOME=/usr/lib64/jvm/java-10-openjdk-10

# Add or update:
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:$JAVA_HOME/lib
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:$JAVA_HOME/lib/server
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/home/user1/modules/endurox-java/libsrc/c:/home/ ↩
    user1/modules/endurox-java/libexjlds
```

FreeBSD:

```
$ vi ~/ndrx_home

Add:


# Java settings
export JAVA_HOME=/usr/local/openjdk8

# Add or update:
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:$JAVA_HOME/jre/lib/amd64
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:$JAVA_HOME/jre/lib/amd64/server
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/home/user1/modules/endurox-java/libsrc/c:/home/ ↩
    user1/modules/endurox-java/libexjlds
```

3.5 Preparing to build and build

Before we start to build, lets load the environment, so that cmake can properly resolve the Java resources (via **JAVA_HOME**).

```
$ . ~/ndrx_home
$ cd endurox-java
$ cmake .
$ make
```

3.6 Enduro/X Java XA Test Configuration

In order to perform testing of Oracle (02_xaoracle) and Posgresql (03_xapostgres) the databases and environment must be configured.

The environment contains host names, users, passwords and database names. The build process will automatically skip these tests, if environment is not configured.

Database configuration (users, environment variables) are configured as part of the building_guide(guides)(Enduro/X Building Guide, Enduro/X basic Environment configuration for HOME directory).

3.7 Configuration of Oracle DB tests

Once the Oracle environment is configured, the test database tables must be created. that could be done in following way (assuming that ~/ndrx_home is properly set):

```
$ source ~/ndrx_home

$ cd endurox-java/tests/02_xaoracle/conf

$ ./sqlplus.run
```

```
SQL> @tables.sql

Table created.
```

After this, system is ready for Oracle DB Unit tests.

3.8 Configuration of Posgresql DB tests

To configure PostgreSQL for Java tests, corresponding database tables for test scenarios must be created. If the environment is properly configured, then table creation can be done in following way:

```
$ source ~/ndrx_home

$ cd endurox-java/tests/03_xapostgres/conf

$ cat tables.sql | ./psql.run
CREATE TABLE
```

Now PosgreSQL database is ready for Enduro/X Java tests.

3.9 Executing the unit tests

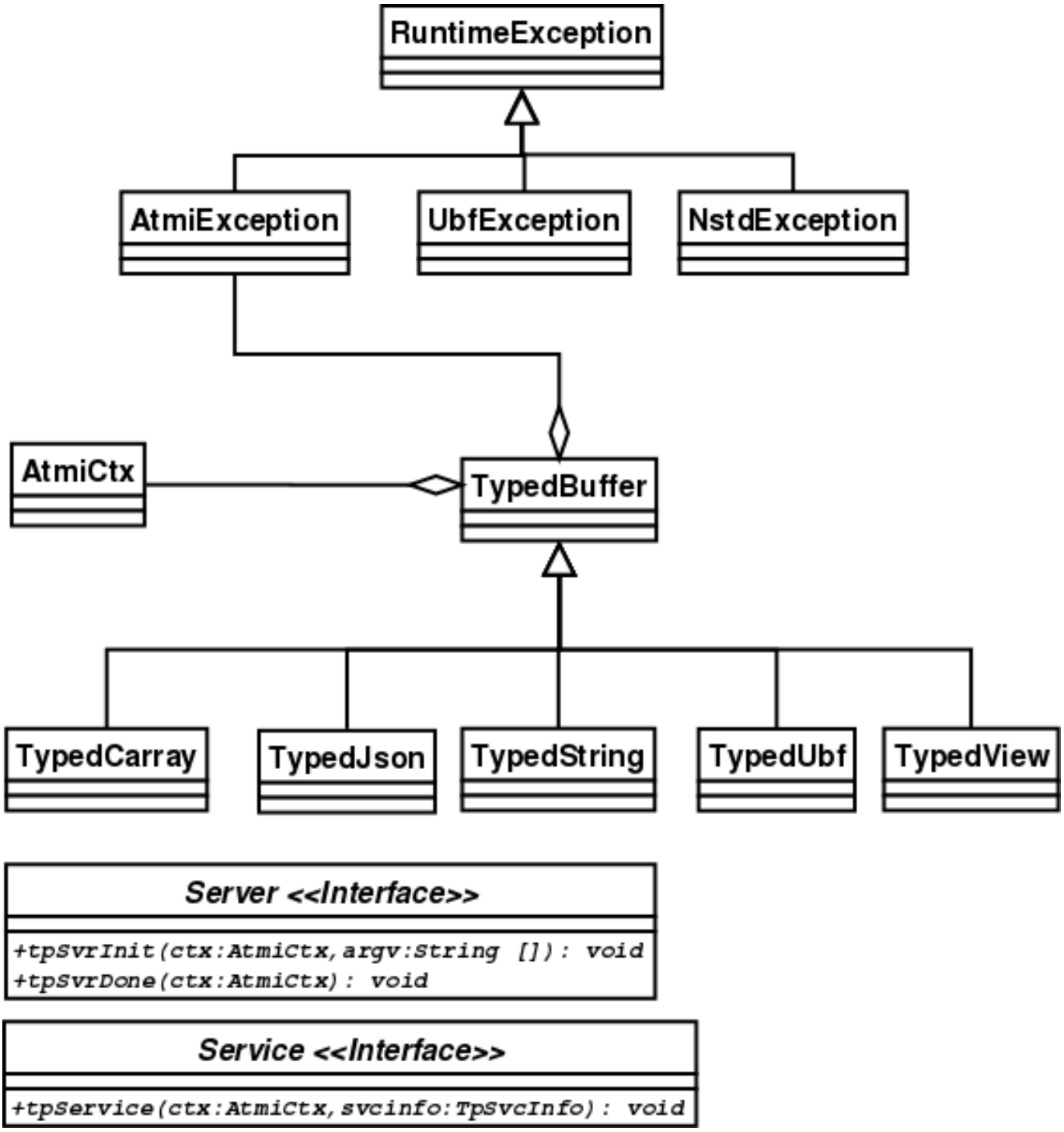
To execute module tests, the environment, database tables, etc shall be created as written before. Once all is ready, the tests can be executed in following way:

```
$ cd endurox-java/tests
$ ./run.sh
```

Chapter 4

General Enduro/X/Java concepts

The object hierarchy is as follows (Class diagram):



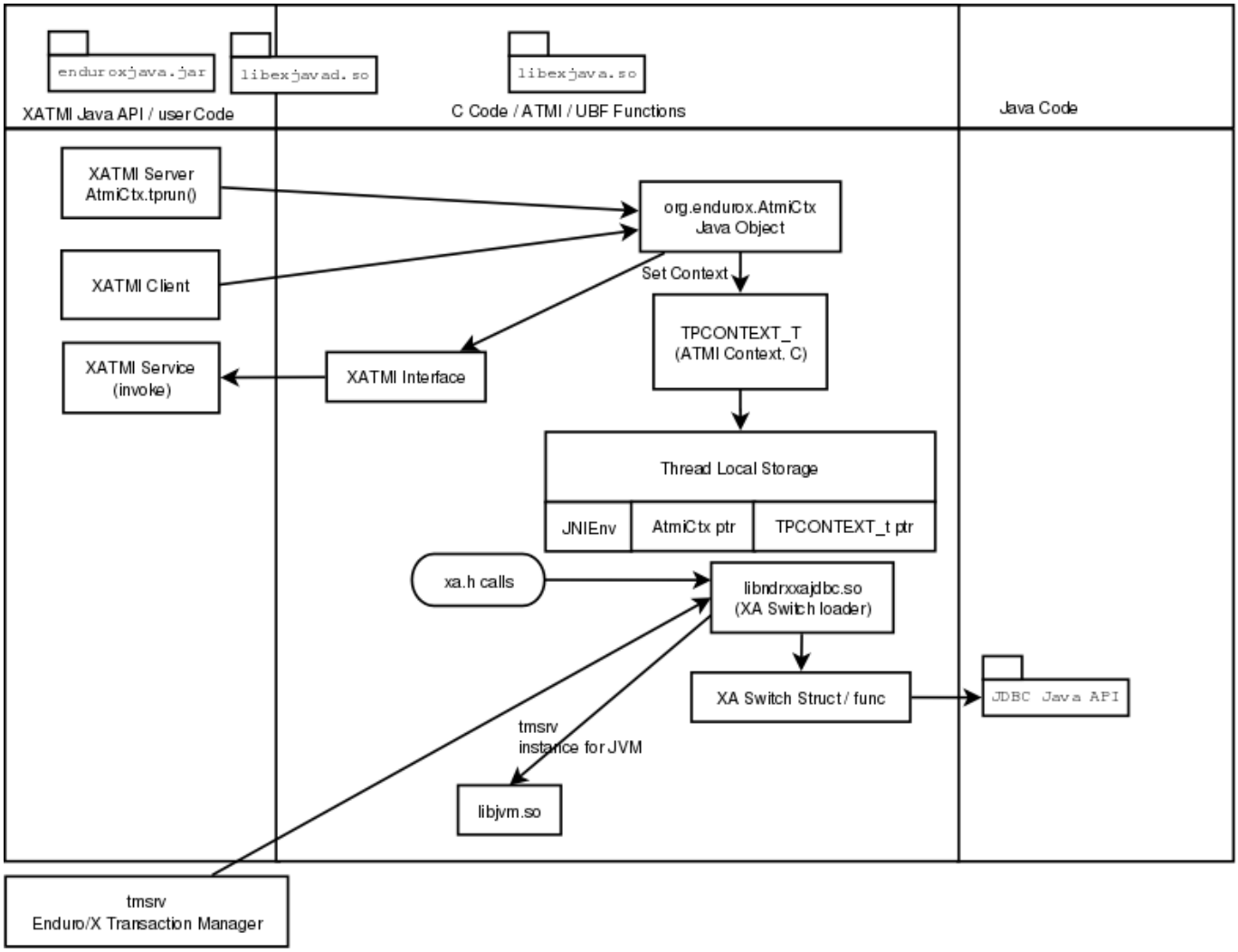
Not all classes are mentioned in this diagram, such as exception and other utility classes. But they key access class to Enduro/X APIs are org.endurox.AtmiCtx. For this class is associated with Enduro/X XATMI client or servers session. Also needs to keep in mind that for one process, there shall be only one XATMI server. Thus AtmiCtx.tprun() shall be called only from single Java thread. Java XATMI client session in turn can be created as much as needed.

In the background of whole java module, following key principles are used:

- All meta data: Classes, Methods and Fields are cached, for performance reasons.
- Enduro/X thread local storage are used for running in C side during Java calls, Special function ndrx_ctx_priv_get() is used to retrieve generic TLS fields where data such as Java env, Java ATMI Context object reference, ptr to self C context.

- When call from Java is made to C and when in turn C calls back Java (for XA and Java XATMI servers) processing, these global variables are used.

Key concepts of the Enduro/X Java package can be seen in following figure:



Chapter 5

Dynamic C libraries

Enduro/X Java C binding code consists of the following libraries:

- libexjava.so - main java Enduro/X binding code. This translates all java attributes from JNI interface to standard XATMI C interface.
- libexjavald.so - this is wrapper library of the libexjava.so. Loaded by java. The wrapper is needed for reason of the way in which java loads the libexjava.so by **System.loadLibrary()**. The symbols are not loaded into global process address space (i.e. with out **RTLD_GLOBAL**). Thus when XA transactions are used, the libndrxxajdbc.so is loaded by Enduro/X which in turn tries to access resources from libexjava.so (which exposes JDBC XA API). This this results in fact that XA API is not visible from such C code. Thus to avoid this, the **libexjavald.so** is introduced which loads the libexjava.so into global address space, and redirects the JNI calls to libexjava.so. The redirect code is generated by **genwrap.pl** script. The script parses the JNI header files to extract the function signatures and generates the corresponding proxy code to **libexjava.so**.
- libndrxxajdbc.so

Enduro/X Java outer classes are supported by C backend which binds the Java calls to actual XATMI C calls. Normally native libraries are loaded **System.loadLibrary()** java method. And it would be epex

Chapter 6

Distributed transaction processing architecture

The nice thing about Java is that their JDBC drivers, are that they provide two phase commit interfaces. The basic principle for the operations are the same which are used by X/Open XA interface. See https://docs.oracle.com/javaee/5/api/javax/transaction/xa/XAResource.html.

Enduro/X by it self uses following architecture for the XA two phase transactions, thus bindings added to Java shall support XA transactions too. There are known "standard" java APIs for this like JTA, but Enduro/X brings as close as possible XATMI API To Java, thus transactions are managed by XATMI API, which basically consists of following methods:

- AtmiCtx.tpopen - Configure resource manager, create instance of XAResource and XAConnection associated with ATMI Context
- AtmiCtx.tpclose - Disconnect from resource manager, delete XAResource and XAConnection associated with ATMI Context
- AtmiCtx.tpbegin - Start the transaction
- AtmiCtx.tpsuspend - Suspend current transaction, put context outside of any transaction
- AtmiCtx.tpresume - Resume suspend transaction, put context back into global transaction
- AtmiCtx.tpcommit - Commit the transaction
- AtmiCtx.tpabort - Abort current transaction
- AtmiCtx.tpgetconn - get connect object from XAConnection. The pooling and closing of connection shall done by programmer.

The transaction management, communications with transaction manager (Enduro/X **tmsrv** binary are performed by Enduro/X C libraries, but due to fact that JDBC drivers live in Java side, the callbacks from C are done back to Java. To get things more complex, Enduro/X uses standard approach of loading XA drivers from C side shared library. Once Enduro/X Core together with Java modules are booted, they are not aware of users willing to use JDBC, in fact Enduro/X Core does not know anything about JDBC. But Enduro/X Java module provides special library named "libndrxxajdbc.so" (our corresponding counter part for MacOS), which in turn expects in "NDRX_XA_RMLIB" (resource managers) configuration parameter expects "libexjava.so" to set. The libexjava.so provides handler to resolve the XA Switch. At startup static XADataSource is initialized. The initialization is done by parsing JSON configuration string found in **NDRX_XA_OPEN_STR**. The syntax for Open String is following

```
{"class":"<JDBC Driver Class Name>",
        "set": {
                "<Set Method Of Class Object 1>":"<Value to bet set 1>"
                ,"<Set Method Of Class Object 2>":"<Value to bet set 2>"
                ,"<Set Method Of Class Object N>":"<Value to bet set N>"
                ,"<Set Method of Properties 1": {
                        "<Property 1 Setting 1>":"<Value to bet set 1/1>"
                        "<Property 1 Setting N>":"<Value to bet set 1/N>"
                    }
                }
        }
```

Thing is that Configuration of XA JDBC Drivers are not standard. There is no standard set of XADataSource methods to configure the driver. Thus Enduro/X uses generic approach to create driver instance and configure it via JSON configuration string. This string accepts:

1. Class name (NOTE! The JDBC driver must be loaded either via linkage or by classpath)
2. A group of set method names and their values. The value types accepted are: **Short**, **Long**, **Integer**, **Byte**, **Float**, **Double**, **Boolean**, **String**. The values for these data types are parsed as strings.
3. An setter method accepting **java.util.Properties**, accepts JSON sub-objects with string values.

And example of XA Open String is following (used by Oracle thin JDBC Driver):

```
[@global/DB1_JDBC]
NDRX_XA_RES_ID=1
NDRX_XA_OPEN_STR={"class":"oracle.jdbc.xa.client.OracleXADataSource",
        "set": {
                "setUser":"${EX_ORA_USER}"
                ,"setPassword":"${EX_ORA_PASS}"
                ,"setURL":"jdbc:oracle:thin:@${EX_ORA_HOST}:${EX_ORA_PORT}/${EX_ORA_SID}"
                ,"setConnectionProperties":{
                        "defaultRowPrefetch":"2"
                        ,"oracle.jdbc.TcpNoDelay":"true"
# Number in milliseconds
                        ,"oracle.jdbc.ReadTimeout":"6000"
                        }
                }
        }
NDRX_XA_CLOSE_STR=${NDRX_XA_OPEN_STR}
NDRX_XA_DRIVERLIB=${NDRX_APPHOME}/../../xadrv/libndrxxajdbc.so
NDRX_XA_RMLIB=${NDRX_APPHOME}/../../libsrc/c/libexjava.so
NDRX_XA_LAZY_INIT=1
```

The XADataSource is configured during the XATMI Startup or during the first XA call (if lazy init is used).

6.1 Transaction Manager operations with JDBC drivers

Enduro/X transaction manager **tmsrv(8)**, is not aware of the Java. The only thing it processes is XA Driver loaded by **NDRX_XA_DRIV** configuration parameter. Which in turn provides the Enduro/X Java binding module **libexjava.so** found in **NDRX_XA_RMLIB**. The JDBC XA library finds out that this is not java which initiated driver loading, thus new Java Virtual Machine instance is created and hosted within tmsrv. VM is configured with settings form [@java] (with CCTAG support) section. Thus there shall be class path configured with -cp or -classpath settings in Java opts. From this class path further the JDBC XA Data Source class is loaded.

Chapter 7

Enduro/X Java Linker

...

Chapter 8

Enduro/X Java XATMI Client process clean shut-down

The standard java shutdown signal handling does not work well in the Enduro/X Java environment, i.e. "Runtime.getRuntime().addShutd
Problem is that java may receive signal at any time at any thread. Even if thread is the Enduro/X C libraries. Such signal can damage the system calls Enduro/X is doing, or this might interrupt/corrupt some java environmental settings at C side, due to executing Java code on the signal arrival. Thus the segmentation faults, etc can be received during such shutdown approach.

To avoid these problems, Enduro/X offers its own mechanisms for receiving the shutdown notifications. The mechanism is to install the runnable object in the C runtime. At the installation time, the signal handlers are re-configured and new thread is standard which monitors the arrival of the following signals:

- **SIGTERM**
- **SIGINT**
- **SIGHUP**

Once any of these signals are received, the java.lang.Runnable callback is executed. Next step is for user application to terminate properly e.g setting some global termination flag or any other mechanism.

To active the shutdown signal monitor thread, use the **org.endurox.AtmiCtx.installTermSigHandler())** static method.

Chapter 9

NetBeans configuration - standard development IDE

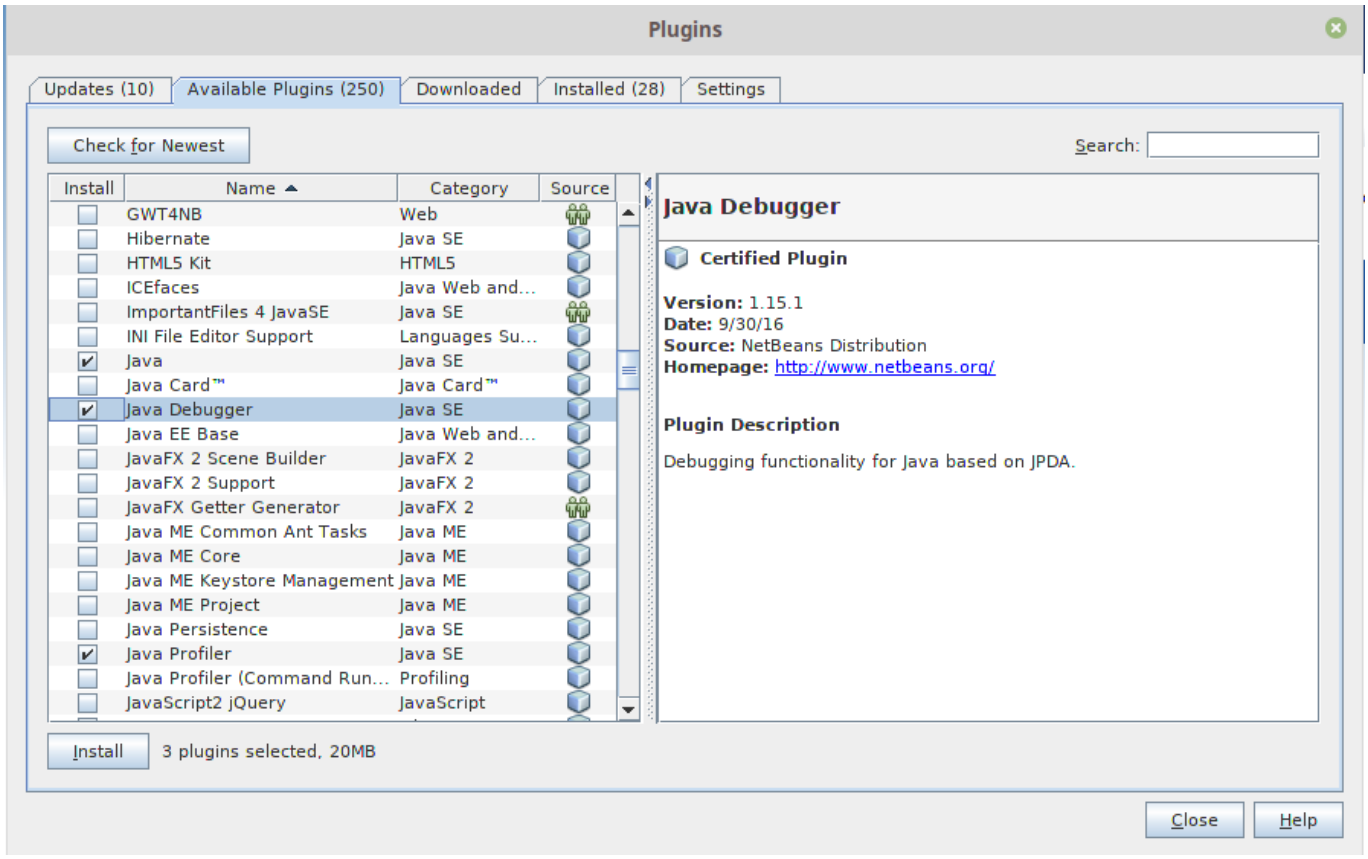
For Enduro/X and other related modules, NetBeans is preferred IDE for development. As module is programmed in Java and C languages, two projects in NetBeans are required. As NetBeans does not allow to project to co-exist in the same folder, some play with symbolic links into separate folder are required. This document will guide you for setting up the environment for developing Enduro/X for Java.

9.1 Packages for Java

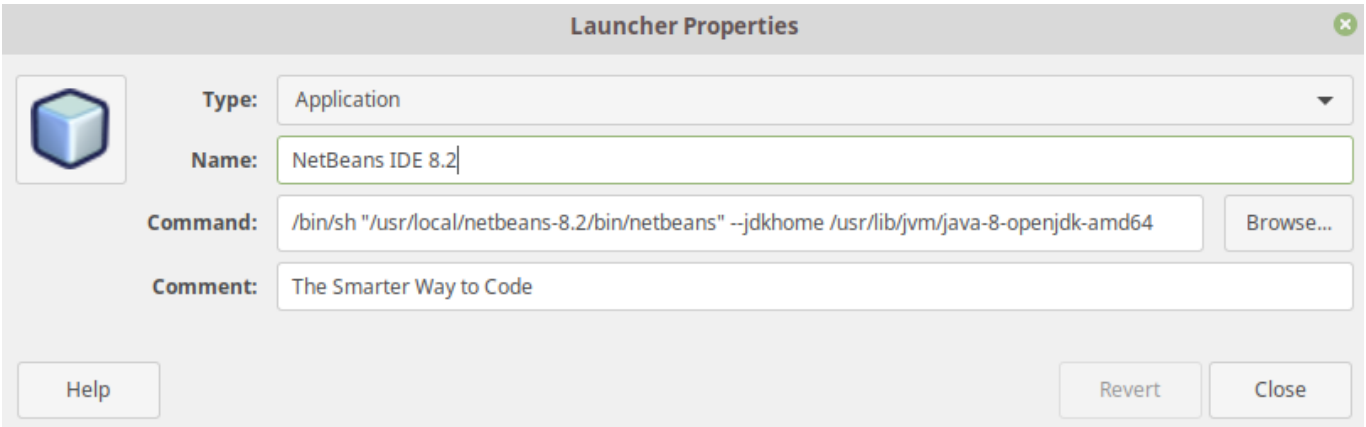
This document assumes that NetBeans for C/C++ are installed. Thus to get Java projects working, following additional plugins must be installed. As plugins require JDK to be present for NetBeans, the IDE must be started with *--jdkhome* attribute. In particular case NetBeans 8.2 was installed on Linux Mint Mate 19 as a root. For this document we will use "java-8-openjdk-amd64".

```
$  /usr/local/netbeans-8.2/bin/netbeans --jdkhome /usr/lib/jvm/java-8-openjdk-amd64
```

Once NetBeans are started, go to: **Tools > Plugins > Available Plugins** and select following ones for install:



Once modules are installed, it is recommended to update the NetBeans launcher shortcut, because the jdkhome argument is mandatory in order to use java projects



9.2 Checking out Enduro/X Java project

With this step we will prepare two folders for the project. The first one is default project folder "endurox-java" checked out from source repository. The second one (which will be actually used by Java part for NetBeans) is created. And symbolic links are added

```
$ mkdir endurox-j
$ cd endurox-j
$ ln -s ../endurox-java/build.xml .
$ ln -s ../endurox-java/tests .
$ ln -s ../endurox-java/libsrc .
```

9.3 Opening projects in NetBeans

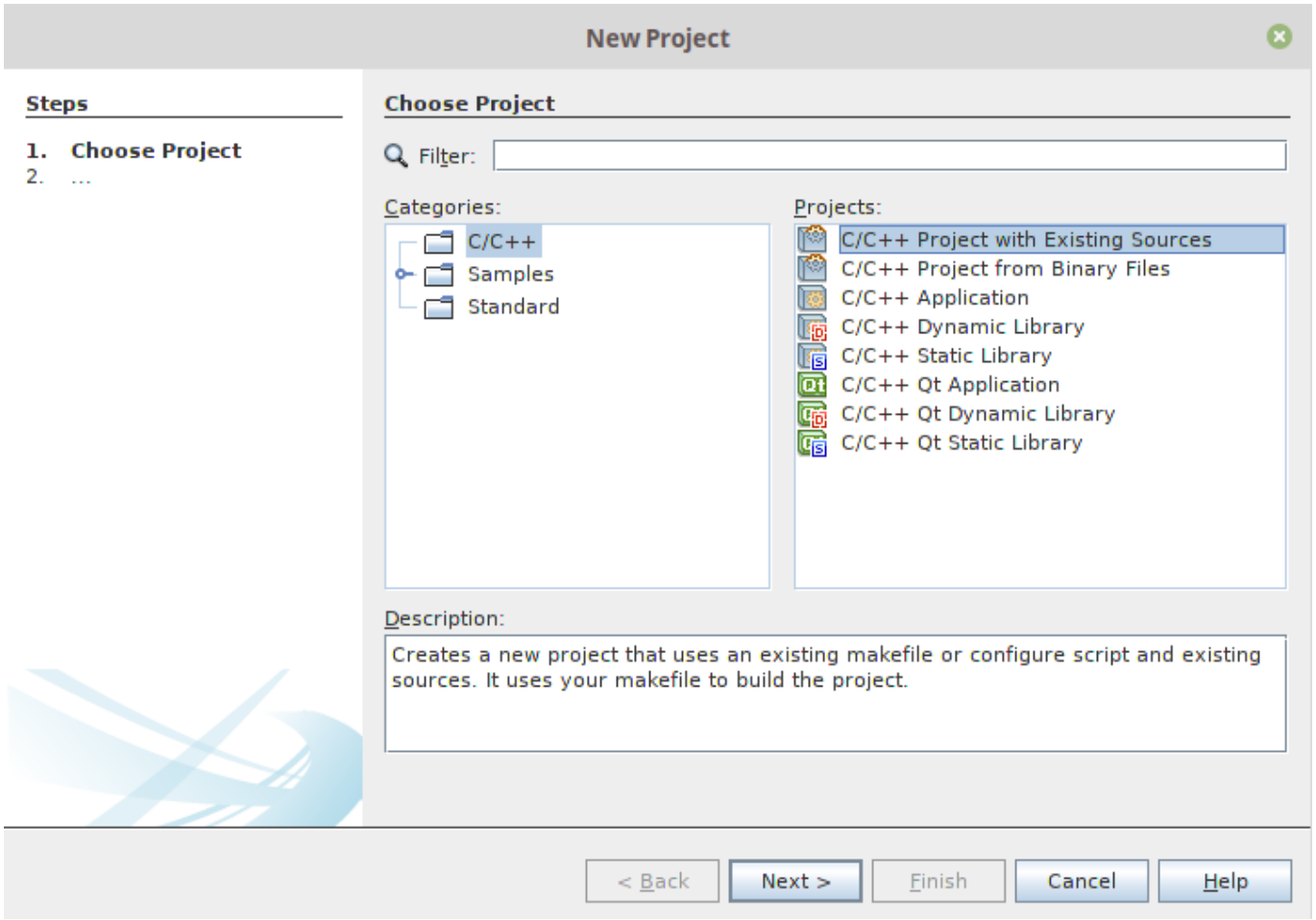
The main project is "endurox-java" which is processed by CMake. The CMake build performs building of all parts Java and C. But for IDE we open this project for as the C project.

9.4 Opening C project in NetBeans

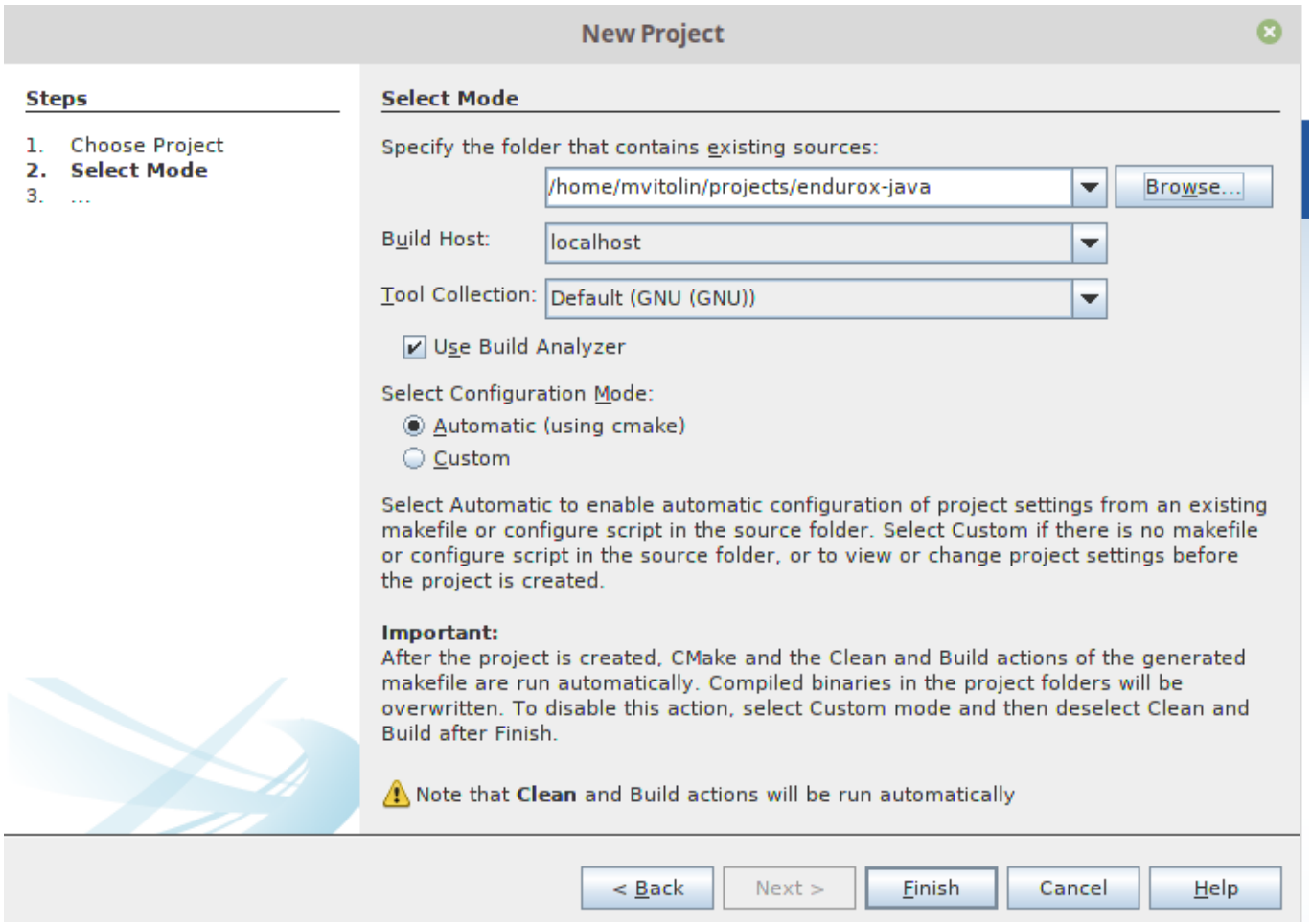
Before opening the project in NetBeans, the cmake shall be run from shell, so that it performs initial configuration, as with NetBeans the configuration is little bit different:

```
$ cd endurox-java
$ cmake .
```

After this step is done, start the NetBeans, and create new project with existing source code:



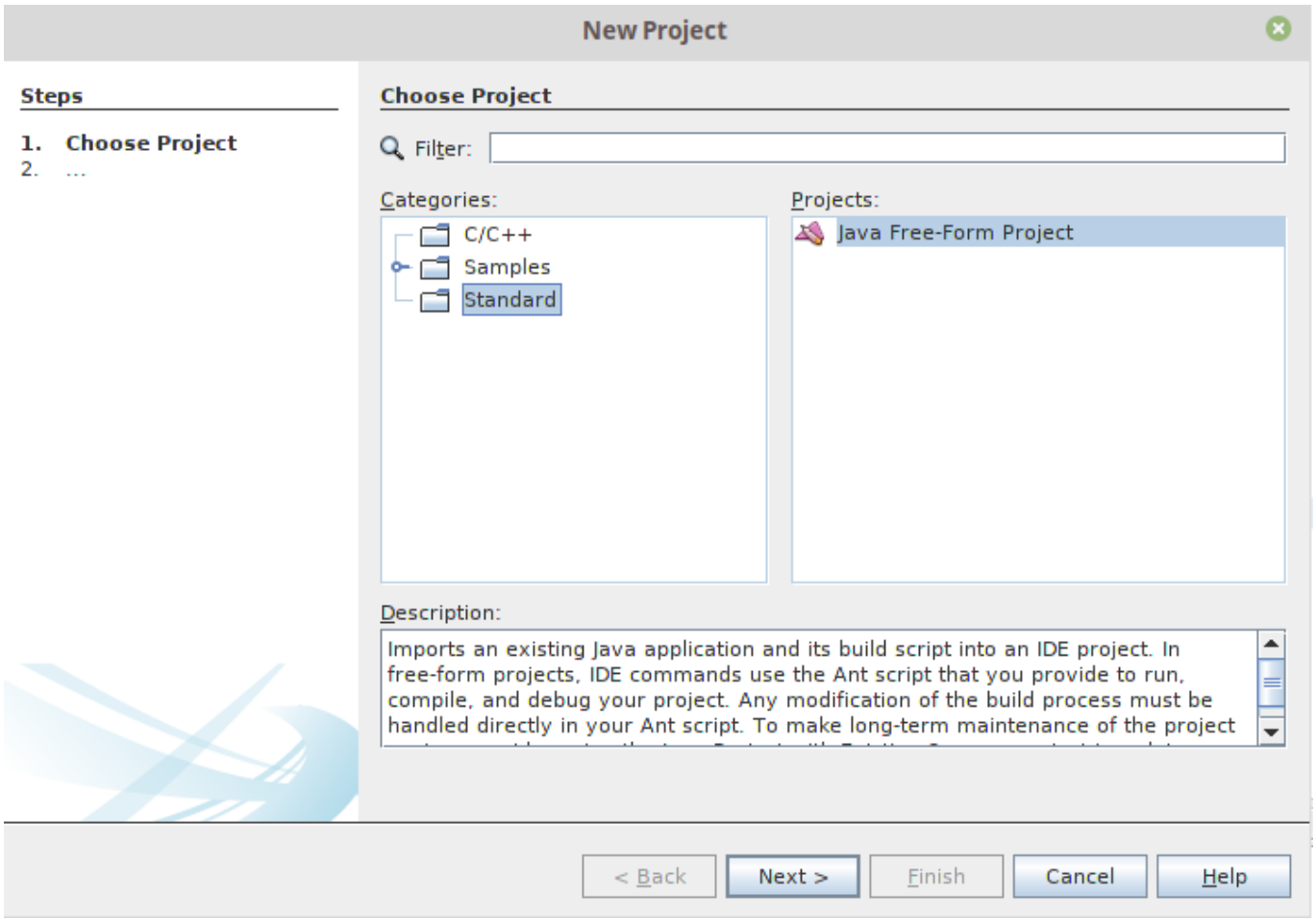
And then select the folder which checked out sources:



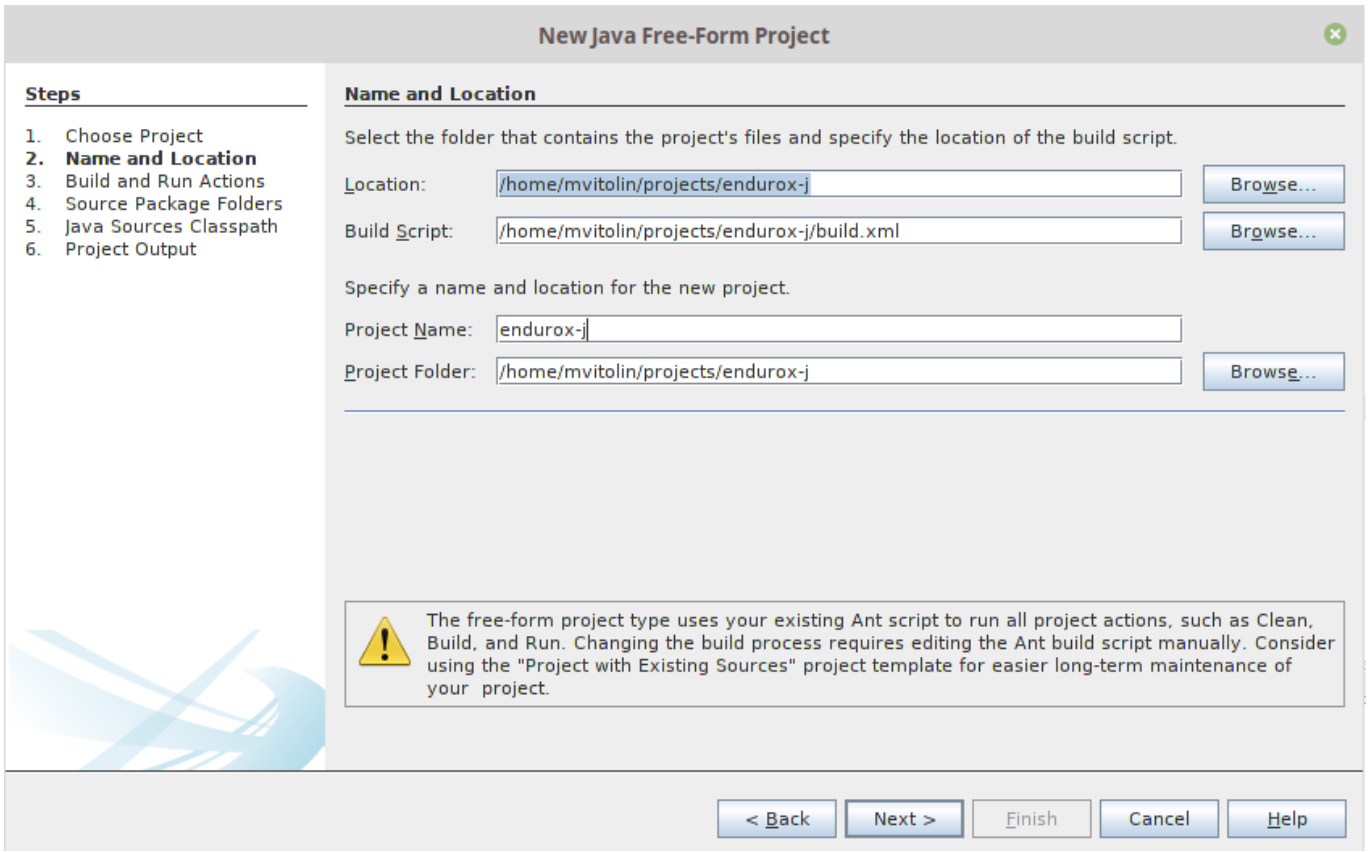
9.5 Opening Java project in NetBeans

The Java project shall be based on folder where symlinks are produced. That is "endurox-j" folder. The project type is "Standard" Java free-form project. The project contains an Ant script which is not normally used for build purposes, but that is used for NetBeans (or Eclipse) to parse the project structure (CMake is not supported yet for Java projects). Also during the development the ant script (endurox-java/build.xml) must be maintained.

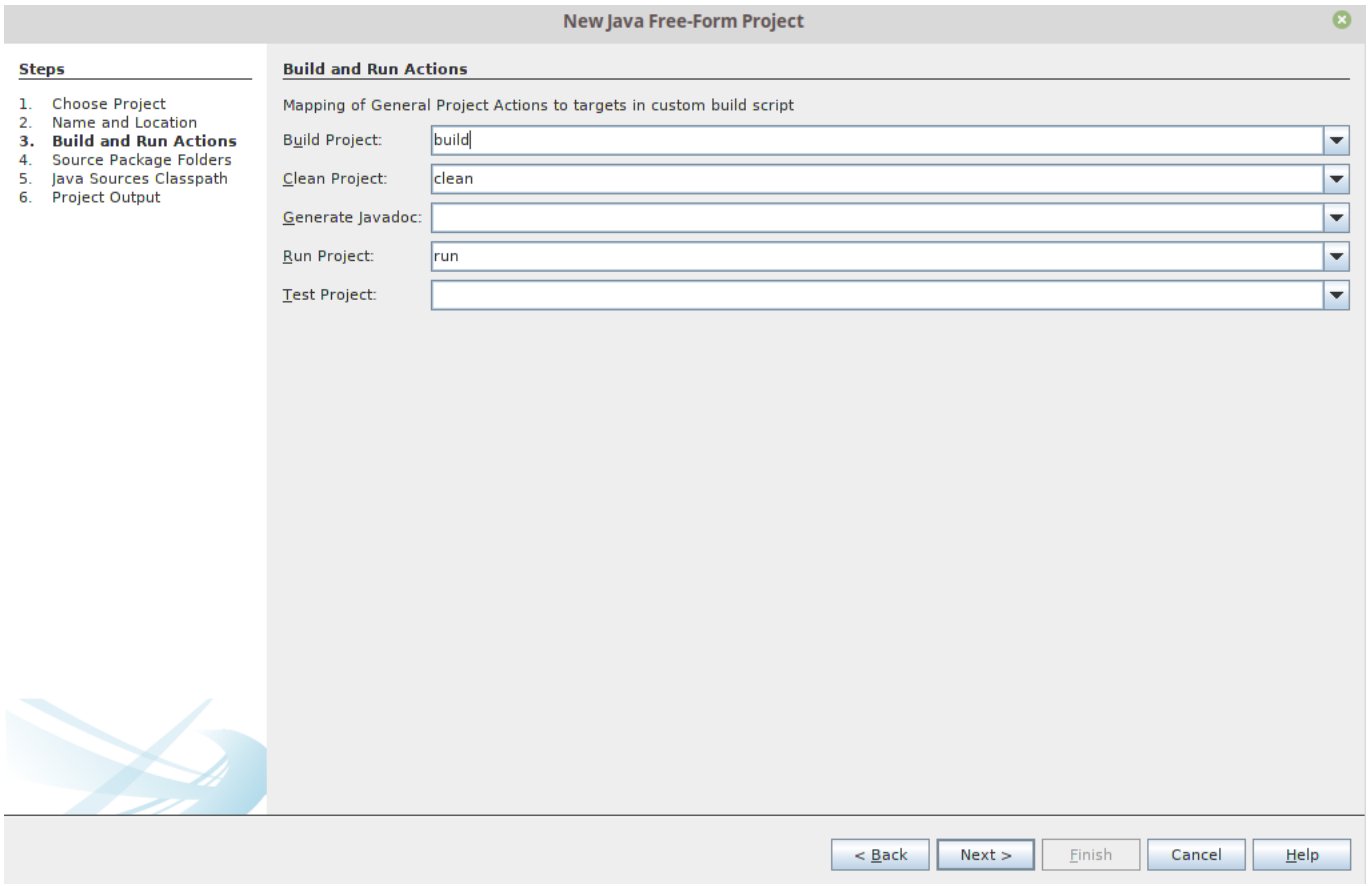
Create a new project:



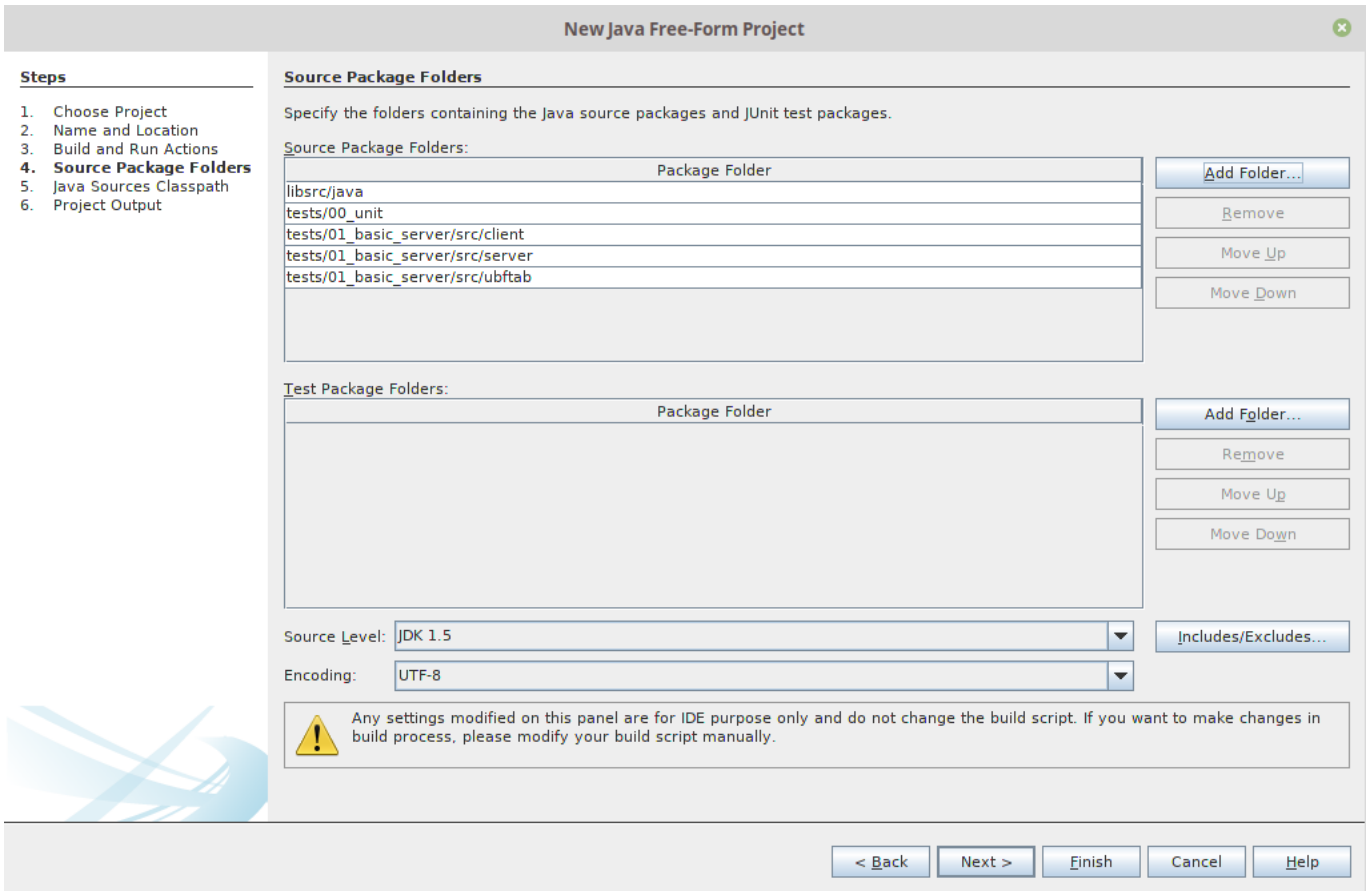
Select project folder:



Ant commands:



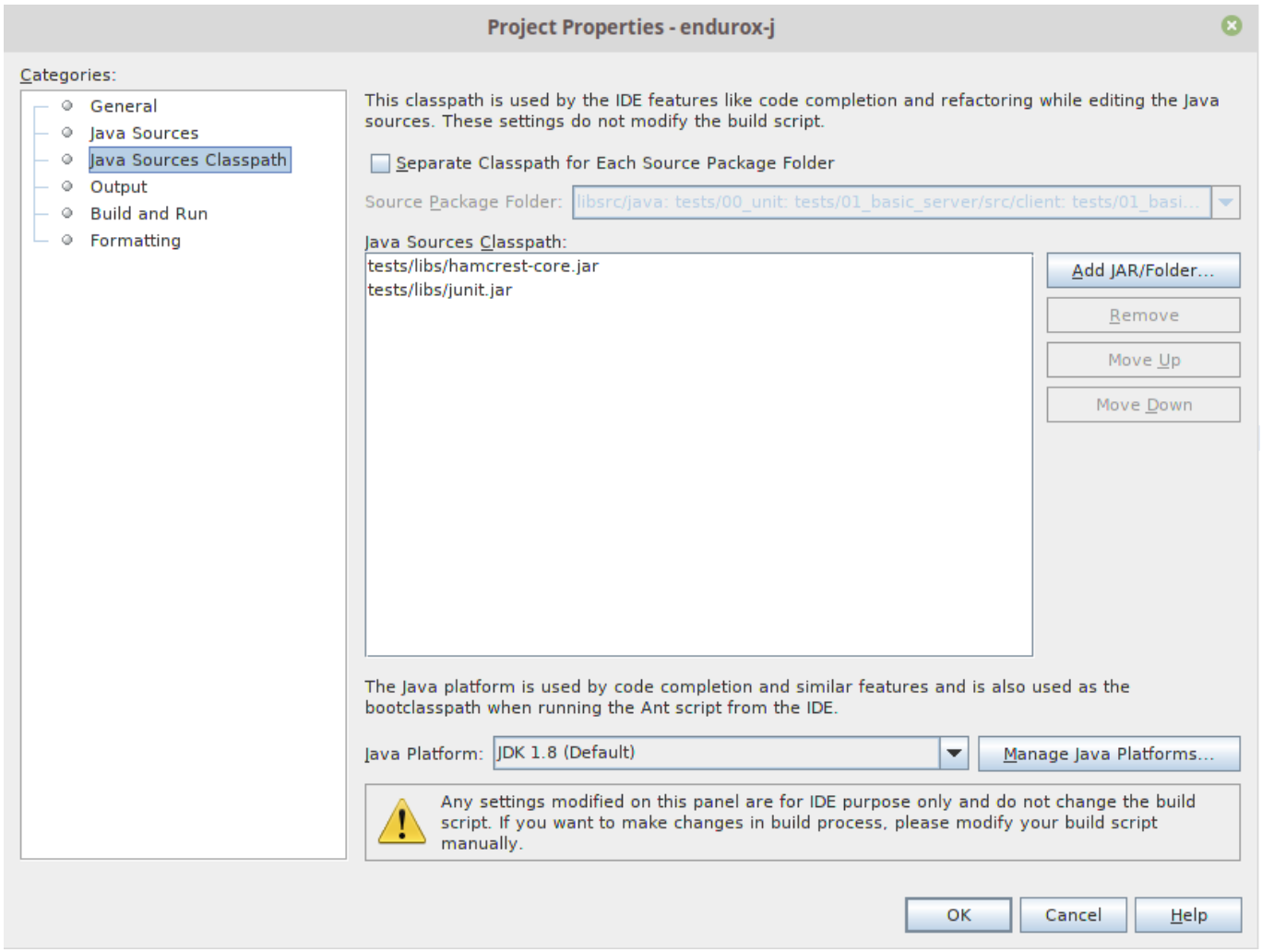
Ant next screen is significant one, as here all Java directories must be manually added, as the libsrc only is added by default. All unit tests which will be changed/added during the development must be added here:



Once project is created, this list can be altered in **project properties > Java Sources**

Also the class path shall include the Junit JARS. The next screen shows how to do it when project is configured, but that can be done during the initial wizard too.

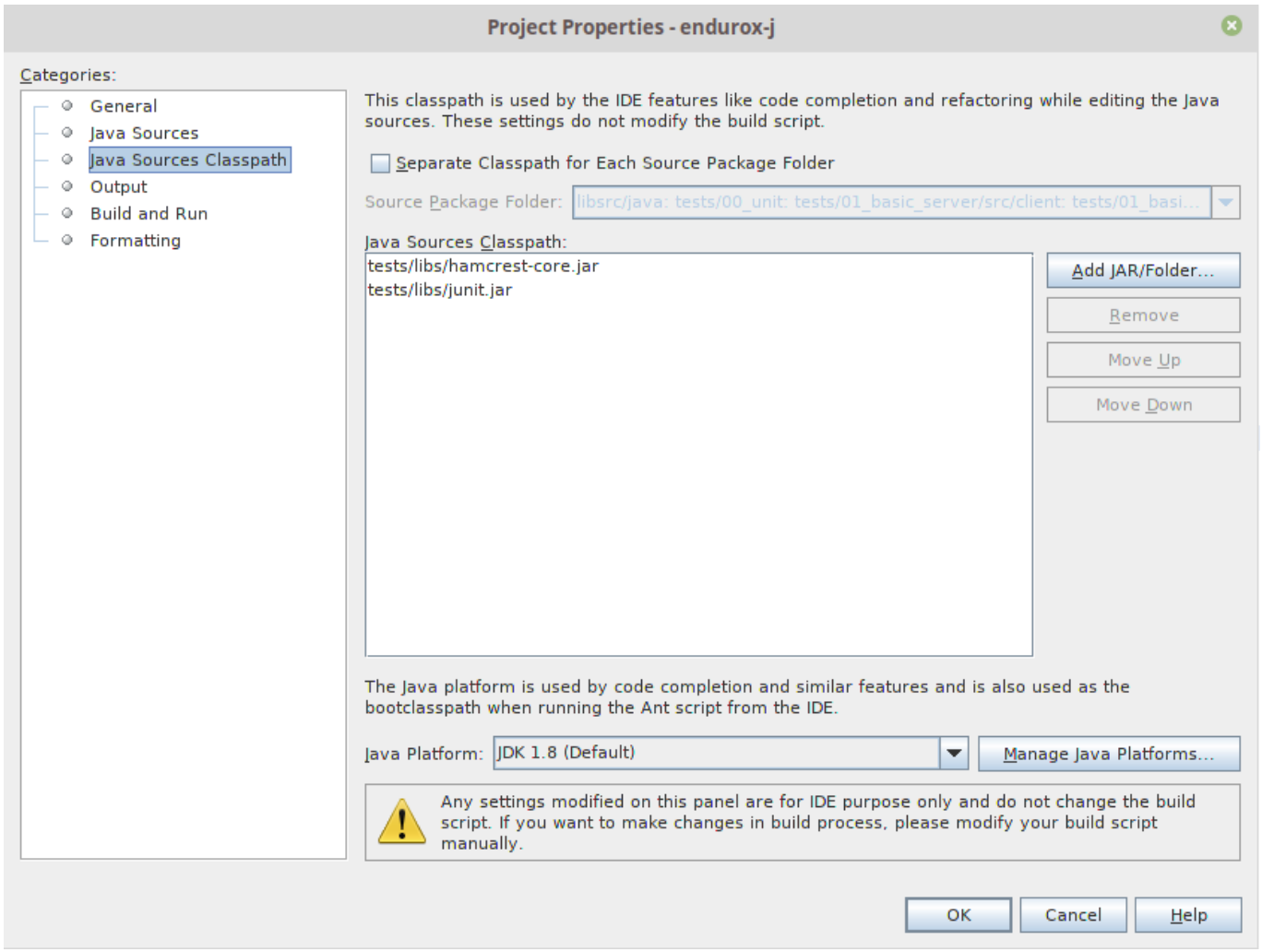
If adding new sources folder get similar message like this (Package folder already used in project):



Then this probably is caused by "endurox-java" C project. There is nothing to do in such case except to go and manually edit the NetBeans project file in

endurox-j/nbproject/project.xml and add the necessary source folders to project, in similar way as other source folders are added.

The class path attributes:


Project Properties - endurox-j
Categories:
General
Java Sources
Java Sources Classpath
Output
Build and Run
Formatting
This classpath is used by the IDE features like code completion and refactoring while editing the Java sources. These settings do not modify the build script.
Separate Classpath for Each Source Package Folder
Source Package Folder:
libsrc/java: tests/00_unit: tests/01_basic_server/src/client: tests/01_basi...
Java Sources Classpath:
tests/libs/hamcrest-core.jar
tests/libs/junit.jar
Add JAR/Folder...
Remove
Move Up
Move Down
The Java platform is used by code completion and similar features and is also used as the bootclasspath when running the Ant script from the IDE.
Java Platform:
JDK 1.8 (Default)
Manage Java Platforms...
Any settings modified on this panel are for IDE purpose only and do not change the build script. If you want to make changes in build process, please modify your build script manually.
OK
Cancel
Help