

iMod SDK

The iMod engine enables adding user protocols with a simple API in JAVA.

Javadoc is available at: <http://imodsdk.techbase.pl>

General information

This document was created for the purpose of describing the method of using available SDK of the iMod application.

The iMod application uses three main channel types:

- Source-Channels
- Access-Channels
- Message-Channels



Independently of the type - implementation of a channel requires the creation of the three main classes:

1. Activator
2. ChannelFactorySrv
3. ChannellImpl



The name of the class is created using the convention [channel_name][channel_type][class_name] where:

- channel_name - any series of characters
- channel_type - the first letter of the channel type (A - Access, S - Source, M - Message)
- class_name - name of the implemented class (Activator, FactorySrv, ChannellImpl)

Activator Class

The Activator Class of the implemented channel must inherit from the CommonActivator class.
Methods requiring implementation: **TemplateMActivator()**

```
public TemplateMActivator(){
    super();
}
```

start()

This method is responsible for the creation and start-up of the ChannelFactorySrv instance.

```
public void start() {
    this.serviceFactory = new TemplateMChannelFactorySrv();
}
```

ChannelFactorySrv Class

The FactorySrv class must implement the ISChannelFactory interface, independently of the channel.
Methods requiring implementation:

IChannel newInstance(IChannelProxy iModEngine, boolean scanmode)

This method is responsible for the creation of a new instance of the implemented channel and its addition to the list of available channels.

```
public IChannel newInstance(IChannelProxy iModEngine, boolean scanmode) {
    IChannel channel = new TemplateMChannelImpl(iModEngine, scanmode);
    if (channel != null)
        this.channelList.add(channel);
    return channel;
}
```

close()

This method is responsible for closing all created instances of the implemented channel.

```
public void start() {
    if (this.channelList.size() > 0) {
        for (IChannel channel : this.channelList)
            channel.close();
    }
}
```

ChannelImpl Class

The ChannelImpl class must implement the IChannel interface, independently of the channel, and inherit from one of the classes depending on the channel type:

- CommonMChannel for message channels

- CommonChannel for access channels
- CommonSChannel for source channels

Methods that are to be implemented:

TemplateMChannelImpl

This method is responsible for starting a channel.

```
public TemplateMChannelImpl(IChannelProxy iModEngine, boolean scanmode) {
    super(iModEngine, scanmode);
    this.setName(this.getChanName());
    this.start();
}
```

getParameter

This method is responsible for servicing a getParameter type query, used by the iMod application to read a given parameter from a channel.

```
public Object getParameter(String id) {
    int new_value = m_generator.nextInt();
    this.m_channel_proxy.setParameter(id, new_value);
    this.m_channel_proxy.setStatus(id, Status.S_OK);
    return new_value;
}
```

You can use following statuses:

- Status.S_OK if read operation succeed.
- Status.S_LAST_READ_FAILED if read operation fails. iMod will try read parameter again.

List<Error> getStatus()

This method requests information on channel status from the iMod application.

```
public List<Error> getStatus() {
    return m_errors;
}
```

void setConfig(IProperties arg0)

This method makes it possible to set additional configuration parameters of the channel.

```
void setConfig(IProperties arg0){
}
```

Status setParameter(String id, Object val)

This method is used by the iMod application to force a change in parameter value in a given channel.

```
public Status setParameter(String id, Object val) {
    this.m_channel_proxy.setParameter(id, val);
    return Status.S_OK;
}
```

You can use following statuses:

- Status.S_OK if write operation succeed.
- Status.S_LAST_WRITE_FAILED if write operation fails. iMod will try set parameter again.

refreshParameters()

This method is used to refresh all available parameters of a given channel (only in source channel).

```
protected void refreshParameters() throws InterruptedException {
    for(String id:this.m_channel_proxy.getParamSpecs().keySet())
        getParameter(id);
}
```

configure()

This method is used for initial channel configuration - for example: in order to bring order to parameters for the purpose of improving output.

```
protected boolean configure() throws IModException {
    m_generator = new Random();
    return super.configure();
}
```

Preparation

For developing custom iMod channel you will always need reference for imodlib.jar library. It can be found in iMod's libraries directory on the device:

- on x500/x500cm3: /home/imod/iModTiger-dev/libs/
- on 9000: /mnt/nand-user/iMod/jar/protocols/
- on x1000:

Following examples uses additionally commons-logging library for logging. It can be found in following directories:

- on x500/x500cm3: /home/imod/iModTiger-dev/libs/
- on 9000: /mnt/nand-user/iMod/jar/libs/
- on x1000:

These libraries can be also found in [iMod SDK Libraries](#) document.

In appendix [iMod SDK Examples](#) there are archives with Eclipse projects of following examples.



Always use imodlib.jar library from your version of iMod

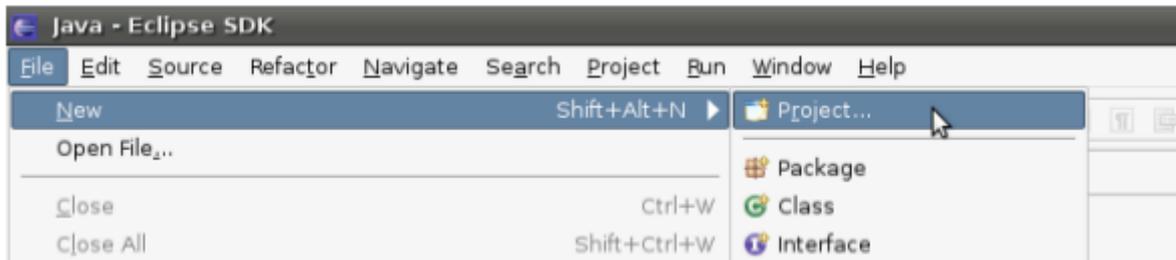
Custom source-channel

Preparation of the Eclipse environment

Section two contains a description of the process for preparing the environment for the purpose of using iMod SDK in the Eclipse IDE environment.

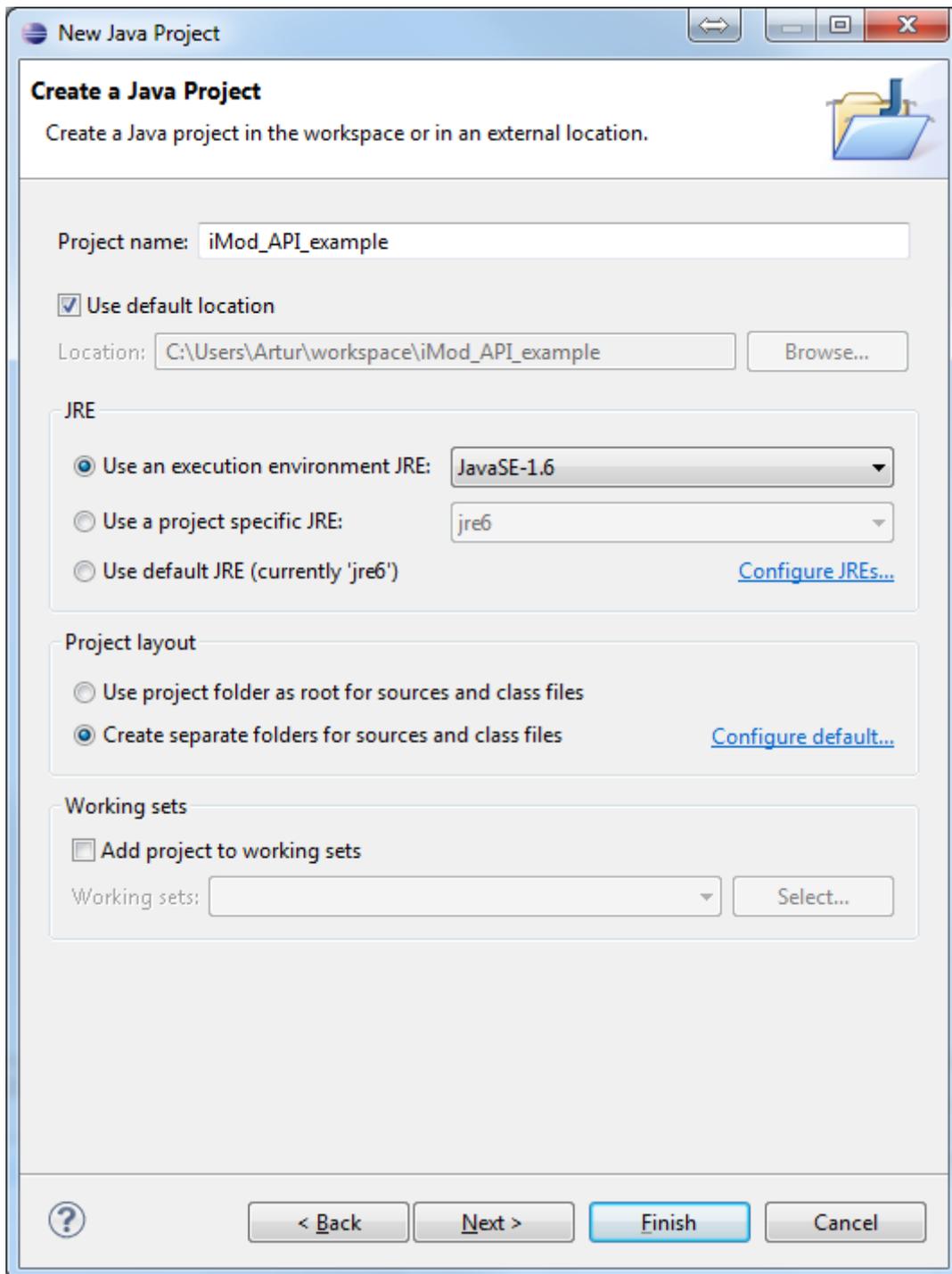
Creating a project

The creation process begins by start-up of the Eclipse program and specification of the path



to the *Workspace*, where created projects will be saved. First, create a project.

In the next window, select the project type: Java Project and click Next. Next, you will be asked to give the project a name - enter iMod_API_example. A path to the directory with your project can be found in the Directory field. To confirm, click Finish.



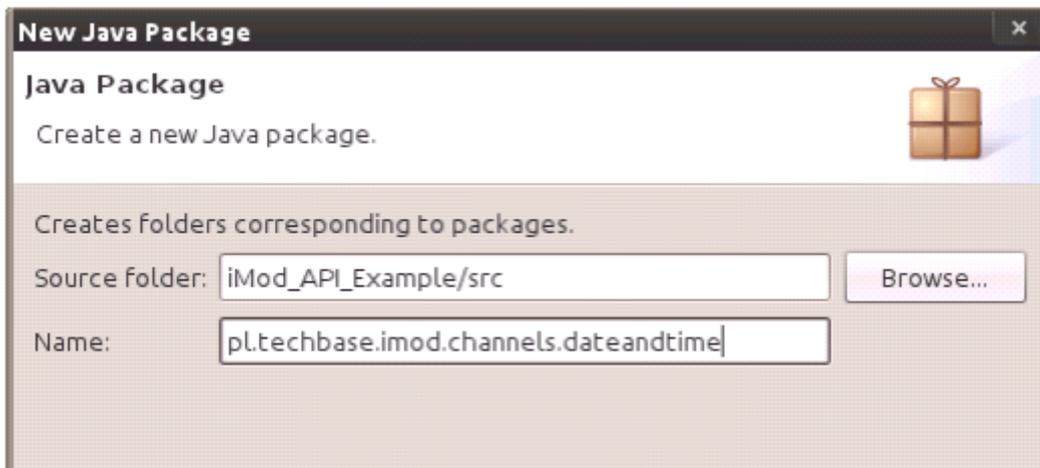
Adding a JAVA PACKAGE

1) Add a package that is compliant with SDK requirements (File new package):



In iMod there already is *DateAndTime* Source Channel. So for use another name i.e. *TIMEANDDATE*. It is important to set custom channel's classes names using CAPITAL LETTERS. Otherwise kernel won't find our custom channel.

```
pl.techbase.imod.channels.name
```



2) Next, in the main project directory, create the lib, subdirectory and copy the imodlib.jar file to it.



The file can be found in the subdirectory /jar/protocols in the iMod application installation directory.

3) Right-click on the added file imodlib.jar and select (Build Path Add to build path)

4) After creating the package and adding the SDK file to the main directory of the project, also add the file build.xml, which will be responsible for compilation and the creation of the jar archive.

Content of the build.xml file, which will be used to create the TIMEANDDDATE channel:

```
<?xml version="1.0" encoding="iso-8859-2"?>
<project name="Example date plugin for iMod - create package" basedir="."
  default="make_jar">
  <property name="techbase.name" value="Customer" />
  <property name="build.dir" location="builder/build/" />
  <property name="class.dir" location="bin/pl" />
  <target name="clean">
    <delete dir="${build.dir}"/>
  </target>
  <!--
  //////////////////////////////////prepare////////////////////////////////////
  -->
  <target name="prepare" depends="clean">
    <mkdir dir="${build.dir}" />
  </target>
  <!-- //////////////////////////////////make_jar//////////////////////////////////// -
  -->
  <target name="make_jar" depends="prepare">
```

```

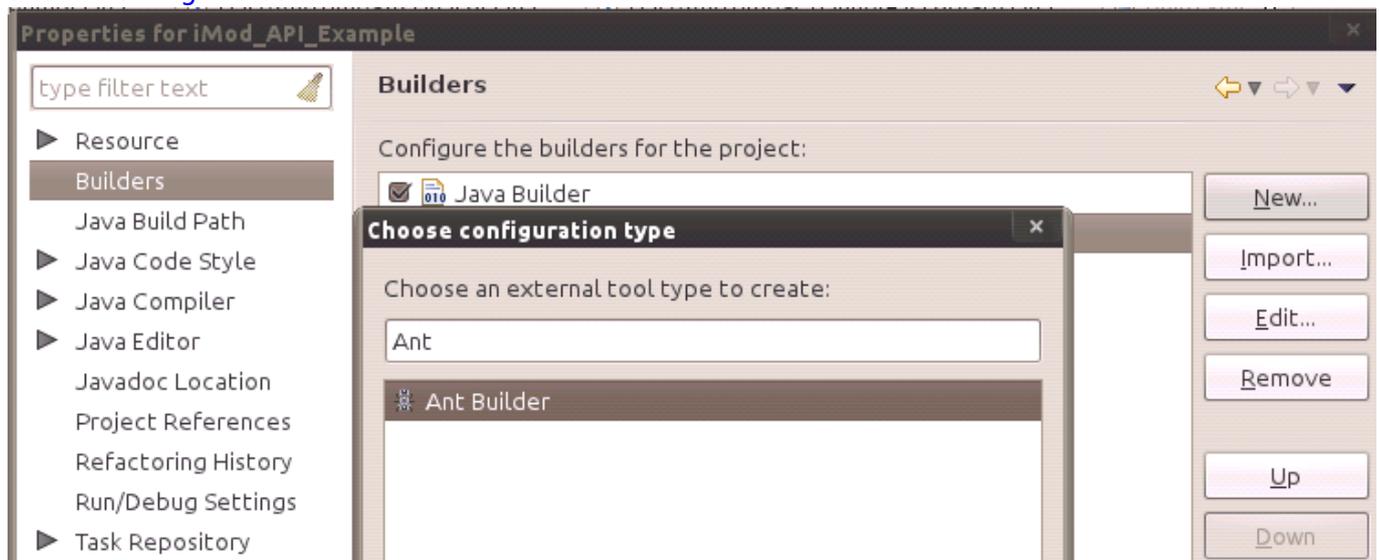
<jar destfile="${build.dir}/TIMEANDDATE.jar">
  <fileset dir="bin">
    <include name="**/*.class"/>
  </fileset>
  <manifest>
    <attribute name="Built-By" value="${techbase.name}"/>
    <attribute name="Class-Path" value="imodlib.jar commons-
logging-1.1.1.jar" />
  </manifest>
</jar>
</target>
</project>

```

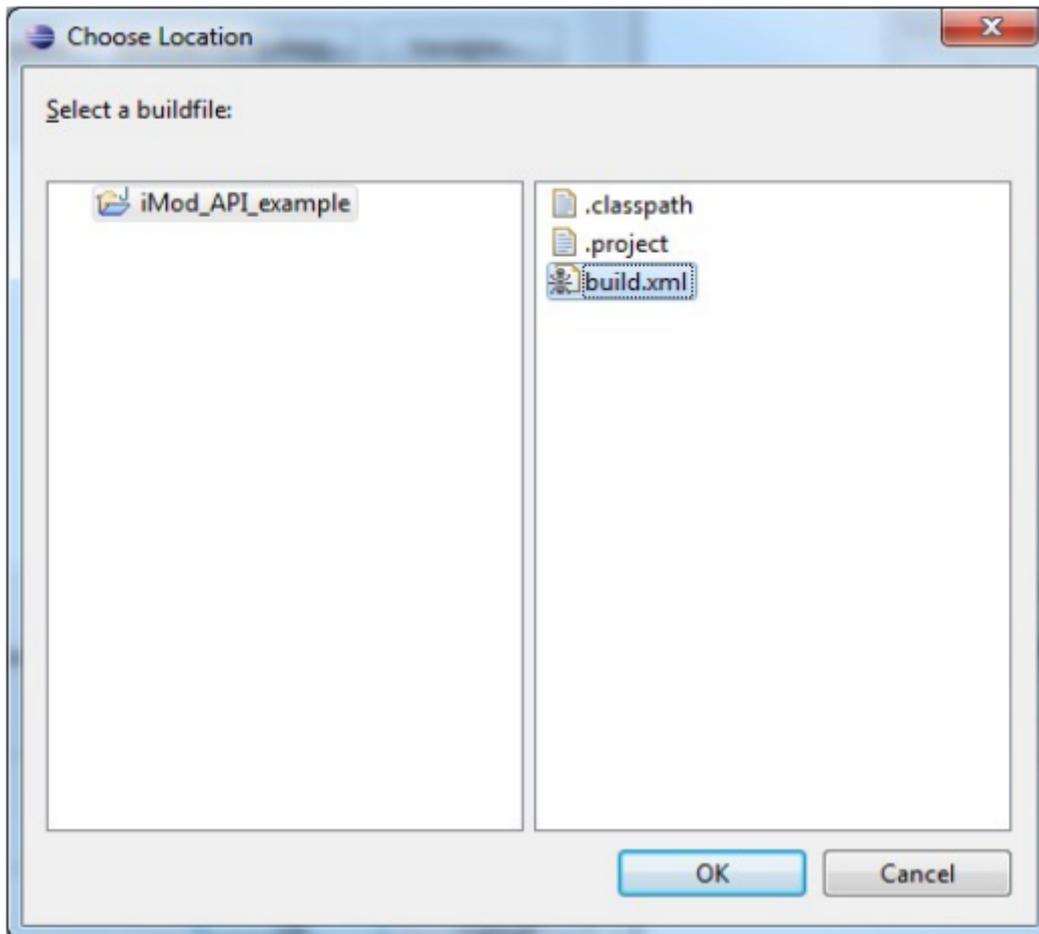


It is important for the `<property name="main.class" value="" />` parameter to indicate a full path to the channel implementing class, e.g. `TIMEANDDATESActivator`

After creating files in the project settings, add a new builder by selecting **Builders>New> Ant Builder** and indicating the `build.xml` file.



After the builder is created correctly, the most important stage can be started, that is, the implementation of an exemplary channel.



Implementation of an exemplary Source-Channel

Section three contains a description of exemplary implementation of communication channels dedicated to SDK of the iMod application. In order to present the structure of a Source-Channel, you will create a channel that enables reading and modification of data on the iMod device.



When building your channel, used libraries from the iMod device. The following libraries will be useful:

- javolution-5.5.1.jar (JAVA package installation directory)
- commons-logging-1.1.1.jar (.../iMod/jar/libs)
- log4j-1.2.16.jar (.../iMod/jar/libs)

For the below example, it is sufficient to additionally import commons-logging and log4j.

For the purposes of this document, the channel will be named: *TIMEANDDATE*.

1) Definition of the structure of the MainConfig.xml file

Begin the implementation process by defining the XML structure that will define the created channel. For the needs of this document, the following XML structure is applied. **Channel definition in the XML file**

```
<source-channel name="NpeDate">
  <protocol name="TIMEANDDATE"/>
  <port>"DayAndTimePort"</port>
  <property name="TIMEANDDATEProperty1" value="true"/>
  <property name="TIMEANDDATEProperty2" value="false"/>
  <cycle>60s</cycle>
</source-channel>
```

Definition of parameters in the XML file

```
<parameter type="int16">
  <id>"Hour"</id>
  <description>"return NPE hour"</description>
  <source-channel channel-name="NpeDate" parameter-id="hour"/>
</parameter>
<parameter type="int16">
  <id>"Minute"</id>
  <description>"return NPE minute"</description>
  <source-channel channel-name="NpeDate" parameter-id="minute"/>
</parameter>
<parameter type="int16">
  <id>"seconds"</id>
  <description>"return NPE seconds"</description>
  <source-channel channel-name="NpeDate" parameter-id="second"/>
</parameter>
<parameter type="int16">
  <id>"Year"</id>
  <description>"return NPE year"</description>
  <source-channel channel-name="NpeDate" parameter-id="year"/>
</parameter>
<parameter type="int16">
  <id>"Month"</id>
  <description>"return NPE Month"</description>
  <source-channel channel-name="NpeDate" parameter-id="month"/>
</parameter>
<parameter type="int16">
  <id>"Day"</id>
  <description>"return num. of day in the month"</description>
  <source-channel channel-name="NpeDate" parameter-id="day"/>
</parameter>
<parameter type="int16">
  <id>"WeekDay"</id>
  <description>"return num. of day in a week"</description>
  <source-channel channel-name="NpeDate" parameter-id="weekday"/>
```

```
</parameter>
```

2) Activator Class

Implementation of a simple channel does not require additional modification of the *TemplateSActivator* class - only the class name must be changed to *TIMEANDDATESActivator*.

Create the class *TIMEANDDATESActivator* with the following content:

```
package pl.techbase.imod.channels.TIMEANDDATE;

import pl.techbase.imod.imodlib.common.CommonActivator;
import pl.techbase.imod.imodlib.plugins.ISChannelFactory;

public class TIMEANDDATESActivator implements CommonActivator {
    public TIMEANDDATESActivator() {
        super();
    }

    @Override
    public ISChannelFactory getFactory() {
        return new TIMEANDDATESChannelFactorySrv();
    }
}
```

3) ChannelFactorySrv Class

Also, the *TIMEANDDATESChannelFactorySrv* class does not require additional modification relative to the *TemplateSChannelFactorySrv* class - with the exception of changes in class names.

Create the *TIMEANDDATESChannelFactorySrv* class with the following content:

```
package pl.techbase.imod.channels.TIMEANDDATE;

import java.util.ArrayList;
import java.util.List;
import pl.techbase.imod.imodlib.engine.IChannelProxy;
import pl.techbase.imod.imodlib.plugins.IChannel;
import pl.techbase.imod.imodlib.plugins.ISChannelFactory;

public class TIMEANDDATESChannelFactorySrv implements ISChannelFactory {

    private List<IChannel> channelList = new ArrayList<IChannel>();

    @Override
    public IChannel newInstance(IChannelProxy iModEngine, boolean scanmode)
    {
        IChannel channel = new TIMEANDDATESChannelImpl(iModEngine,
scanmode);
        if (channel != null)

```

```

        this.channelList.add(channel);
        return channel;
    }

    @Override
    public void close() {
        if (this.channelList.size() > 0) {
            for (IChannel channel : this.channelList)
                channel.close();
        }
    }
}

```

4) ChannelImpl Class

The target channel implementation is realized in the *TIMEANDDATESChannelImpl* class, which gives iModEngine access to the resources of a given source. Create the class *TIMEANDDATESChannelImpl* with the following content:

```

package pl.techbase.imod.channels.timeanddate;

import java.util.ArrayList;
import java.util.Calendar;
import java.util.Iterator;
import java.util.List;
import java.util.Map;
import java.util.Properties;

import org.apache.commons.logging.Log;
import org.apache.commons.logging.LogFactory;

import pl.techbase.imod.imodlib.common.CommonSChannel;
import pl.techbase.imod.imodlib.common.IModException;
import pl.techbase.imod.imodlib.engine.IChannelProxy;
import pl.techbase.imod.imodlib.engine.IData.Status;
import pl.techbase.imod.imodlib.engine.IProperties;
import pl.techbase.imod.imodlib.plugins.IChannel;
import pl.techbase.imod.imodlib.engine.Error;

public class TIMEANDDATESChannelImpl extends CommonSChannel implements
IChannel {
    private List<Error> m_errors = new ArrayList<Error>();
    List<Error> errors;
    private Calendar cal = null;
    private static Log log =
LogFactory.getLog(TIMEANDDATESChannelImpl.class);
}

```



The LOG class will not be visible until the log4j file is imported.

TIMEANDDATESChannelImpl(IChannelProxy iModEngine, boolean scanmode)

The TIMEANDDATESChannelImpl constructor is responsible for starting the channel and additionally, contains information entered into the application log for the purpose of verifying correct class start-up.

Add the following method:

```
public TIMEANDDATESChannelImpl(IChannelProxy iModEngine, boolean scanmode)
{
    super(iModEngine, scanmode);
    this.setName(this.getChanName());
    log.info("Starting TimeAndDate Source Channel Plugin ...");
    this.start();
}
```

getParameter(String id)

The *getParameter* method is responsible for servicing *getParameter* type queries, used by the iMod application for reading a given parameter from a channel. It is important for the method to return a read value using return.

In addition, the internal iMod base must be updated by the read value using the *this.m_channel_proxy.setParameter(id, new_value)* method and must then return information on parameter status to iModEngine using the *setStatus>* method In the created example, the method should contain a mechanism for differentiating the attribute type and should return the date component in the device.

You can use following statuses:

- *Status.S_OK* if read operation succeed.
- *Status.S_LAST_READ_FAILED* if read operation fails. iMod will try read parameter again.

Add the following code to the class:

```
public Object getParameter(String id) {
    int new_value = 0;
    log.info("Try to getParameter: " + id);
    cal = Calendar.getInstance();
    Map<String, IProperties> paramSpecs =
this.m_channel_proxy.getParamSpecs();
    if (paramSpecs.get(id) != null &&
paramSpecs.get(id).keySet().contains("source-channel.parameter-id")) {
        String value =
paramSpecs.get(id).get("source-channel.parameter-id").get(0);
        log.info("getParameter(" + id + ") value: " + value);
        if (value.equals("hour")) {
            new_value = cal.get(Calendar.HOUR);

```

```

    } else if (value.equals("minute")) {
        new_value = cal.get(Calendar.MINUTE);
    } else if (value.equals("second")) {
        new_value = cal.get(Calendar.SECOND);
    } else if (value.equals("year")) {
        new_value = cal.get(Calendar.YEAR);
    } else if (value.equals("month")) {
        new_value = cal.get(Calendar.MONTH) + 1;
    } else if (value.equals("day")) {
        new_value = cal.get(Calendar.DATE);
    } else if (value.equals("weekday")) {
        new_value = cal.get(Calendar.DAY_OF_WEEK);
    } else {
        log.error("Wrong parameter-id!!");
    }
}
log.info("Parameter: " + id + " has value: " + new_value);
this.m_channel_proxy.setStatus(id, Status.S_OK);
this.m_channel_proxy.setParameter(id, new_value);
return new_value;
}

```

getStatus()

The *getStatus* method returns channel status information to the iMod application. Add it to the class:

```

public List<Error> getStatus() {
    return m_errors;
}

```

setConfig(IProperties arg0)

setConfig is the method enabling the configuration of additional channel configuration parameters:

```

public void setConfig(IProperties p) {
}

```

setParameter(String id, Object val)

The *setParameter* method is used by the iMod application to force changes in the parameter value of a given channel.

You can use following statuses:

- Status.S_OK if write operation succeed.
- Status.S_LAST_WRITE_FAILED if write operation fails. iMod will try set parameter again.

Add the following content to the class:

```

public Status setParameter(String id, Object val) {
    Map<String, IProperties> paramSpecs =
this.m_channel_proxy.getParamSpecs();
    log.info("Try to setParameter: " + id + " value: " + val);
}

```

```

    if
    (paramSpecs.get(id).keySet().contains("source-channel.parameter-id")) {
        String value =
paramSpecs.get(id).get("source-channel.parameter-id").get(0);
        log.info("parameter Type: " + value);
        if (value.equals("hour")) {
            cal.set(Calendar.HOUR, (Integer) val);
        } else if (value.equals("minute")) {
            cal.set(Calendar.MINUTE, (Integer) val);
        } else if (value.equals("second")) {
            cal.set(Calendar.SECOND, (Integer) val);
        } else if (value.equals("year")) {
            cal.set(Calendar.YEAR, (Integer) val);
        } else if (value.equals("month")) {
            cal.set(Calendar.MONTH, (Integer) val);
        } else if (value.equals("day")) {
            cal.set(Calendar.DATE, (Integer) val);
        } else if (value.equals("weekday")) {
            cal.set(Calendar.DAY_OF_WEEK, (Integer) val);
        } else {
            log.error("Wrong parameter-id!!");
        }
    }
    log.info("Parameter: " + id + " has new value: " + val);
    this.m_channel_proxy.setParameter(id, val);
    return Status.S_OK;
}

```

refreshParameters()

The method is used to refresh all defined parameters assigned to a given channel. In the case of a simple channel, the refreshParameters method will contain a loop reading individual parameters using the *getParameter(id)* method

```

protected void refreshParameters() throws InterruptedException {
    for (String id : this.m_channel_proxy.getParamSpecs().keySet())
        getParameter(id);
}

```

configure()

The configure method is used for initial channel configuration. In the case of the TIMEANDDATE channel, this method will be responsible for singling out the appropriate initial parameters from the XML file and for bringing order to the parameters for the purpose of improving output.

The configure method is run once during channel initialization – it would be best to include any scalings and other parameter adjustments here – including adding parameters to the list so that, for example, parameters are read immediately after the refreshParameters() method is started – this will improve output – reading speed.

An exemplary definition of a channel can be found in the *MainConfig.xml* file:

```
<source-channel name="NpeDate">
  <protocol name="TIMEANDDATE"/>
  <port>"DayAndTimePort"</port>
  <property name="TIMEANDDATEProperty1" value="true"/>
  <property name="TIMEANDDATEProperty2" value="false"/>
  <cycle>60s</cycle>
</source-channel>
```



For the purpose of showing the method of singling out property property type parameters from the channel definition, *TIMEANDDATEProperty1* and *TIMEANDDATEProperty2* have been added, and their values have been added to the iMod application log.

Add an exemplary implementation of the *configure()* method:

```
protected boolean configure() throws IModException {
    if (super.configure()) {
        Iterator<java.util.Properties> iter = this.m_channel_proxy
            .getConfig().iterator();
        while (iter.hasNext()) {
            Properties prop = iter.next();
            Iterator<Object> iterKey = prop.keySet().iterator();
            while (iterKey.hasNext()) {
                String nextKey = (String) iterKey.next();
                String nextValue = (String) prop.get(nextKey);
                if (nextKey.equals("port")) {
                    log.info("Port recognized as " + nextValue);
                } else if (nextKey.equals("property.TIMEANDDATEProperty1"))
                {
                    log.info("TIMEANDDATEProperty1 has value: " +
nextValue);
                } else if (nextKey.equals("property.TIMEANDDATEProperty2"))
                {
                    log.info("TIMEANDDATEProperty2 has value: " +
nextValue);
                }
            }
        }
    }
    return true;
}
```

Using the *configure()* method, you can acquire and browse the parsed content of the *MainConfig.xml* file.

`super.configure()` method can parse following source-channel specific parameters: gap, cycle, delay, write-delay.

`configure()` method can return *true* - if configuration process ends with success or *false* - if something fails. If `configure()` returns *false*, iMod will call this method again.

You can check the name of the key using the method `nextKey.equals("name")` method:

```
String nextKey = (String)iterKey.next();
String nextValue = (String) prop.get(nextKey);
if (nextKey.equals("port")) {
    (..) /* Support of a given attribute */
}
```

If, for example, `<port></port>` is encountered, then the string of characters entered between the port keys markers can be acquired using the `prop.get(nextKey)` method.

Next, depending on the channel requirements - the entered string is to be interpreted appropriately. For example, thanks to Pattern and Matcher interesting information can be checked and singled out, e.g. the TCP or serial port number and other information such as Baudrate etc.



Character strings entered in markers defining a channel must be appropriately defined and interpreted in reference to a given channel.

5) Deploying on device

- Deploying on x500/x500cm3 device

After concluding correct implementation of the TIMEANDDATE channel and creation of the JAR archive put archive to `/home/imod/iModTiger-dev/libs/` directory and restart iMod.

- Deploying on 9000/x1000 device

After concluding correct implementation of the TIMEANDDATE channel and creation of the JAR archive, modify the iModEngine manifest so that the application kernel can see the class of the new channel.

The first step is to acquire `imodTiger.jar` files from the device. It is easier to add the file to the main project directory in Eclipse.



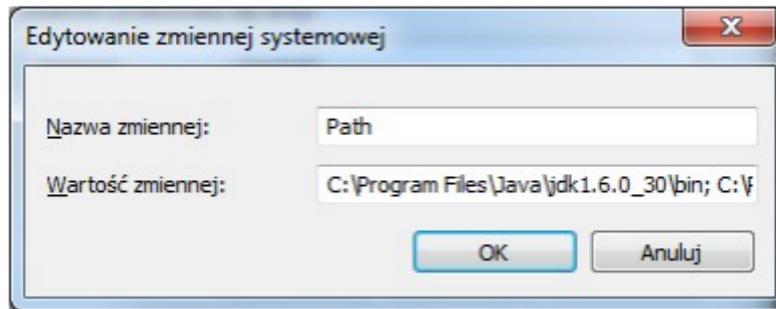
The `imodTiger.jar` file can be found in the main installation directory of the iMod application. The target location can be checked using the command: `getenv IMOD_VERSION`

After downloading the file, unpack the manifest file using the command:

```
jar xf imodTiger.jar META-INF/MANIFEST.MF
```



In the Windows 7 system, if the JAR command is not visible, then a link to the /bin directory must be added at the beginning of the system variable.



Next add an entry to the Class-Path line with the target location of the *date.jar* file:

```
Manifest-Version: 1.0
Ant-Version: Apache Ant 1.7.1
Created-By: 20.1-b02 (Sun Microsystems Inc.)
Built-By: TechBase Sp. z o.o.
Main-Class: pl.techbase.imod.Main
Class-Path: ../jar/libs/snmp4j-1.11.1.jar ../jar/libs/sqlitejdbc-v056.jar
../jar/libs/commons-logging-1.1.1.jar ../jar/libs/log4j-1.2.16.jar
../jar/protocols/imodlib.jar ../jar/protocols/modbus.jar
../jar/protocols/mbus.jar ../jar/protocols/ecl300.jar
../jar/protocols/onewire.jar ../jar/protocols/snmp.jar
../jar/libs/mail.jar ../jar/libs/smslib.jar ../jar/libs/jowfsclient.jar
../jar/libs/slf4j-api-1.5.8.jar ../jar/libs/slf4j-log4j12-1.5.8.jar
../jar/libs/protobuf-java-2.4.1-lite.jar ../jar/libs/zmq.jar
../jar/libs/cron4j-2.2.4.jar ../jar/libs/postgresql-8.4-703.jdbc4.jar
../jar/protocols/date.jar
```

Next, using the following command, update the manifest in the imodTiger file:

```
jar umf META-INF/MANIFEST.MF imodTiger.jar
```

After finishing the update, the new imodTiger.jar file is to be saved to the main iMod application directory on the NPE device.

Save the newly created date.jar file implementing the TIMEANDDATE channel to the directory indicated in the Manifest (jar/protocols).

After saving files, the parameters defined for the new channel can be added to the MainConfig.xml file for the purpose of checking the correctness of operation.

Custom message-channel

In this instruction we will add simple message channel to imod using iModSDK. Our example channel will save incoming messages to files, which pathes are defined as recipients.

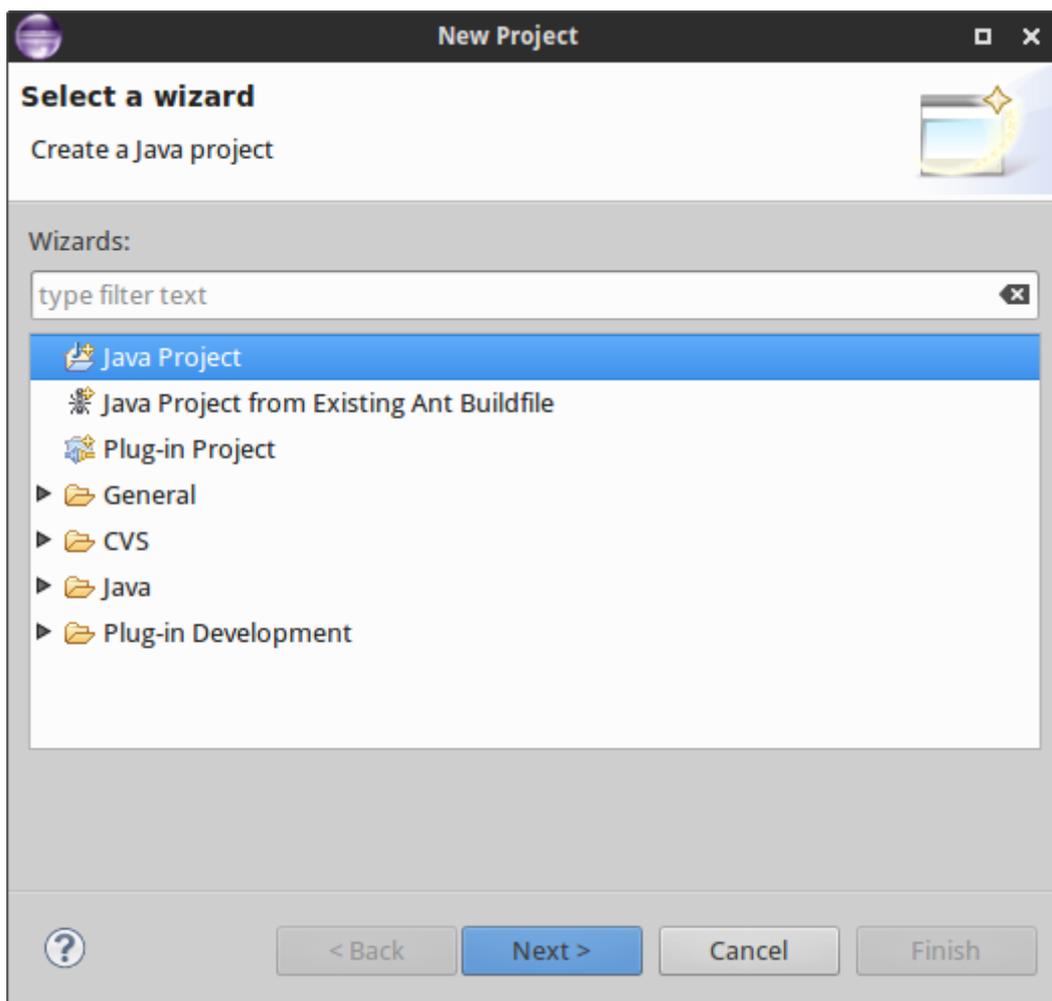


To build custom message channel you will need also javolution-5.5.1.jar library. It can be found in the same directory as commons-logging-1.1.1.jar or in [iMod SDK Libraries](#) page

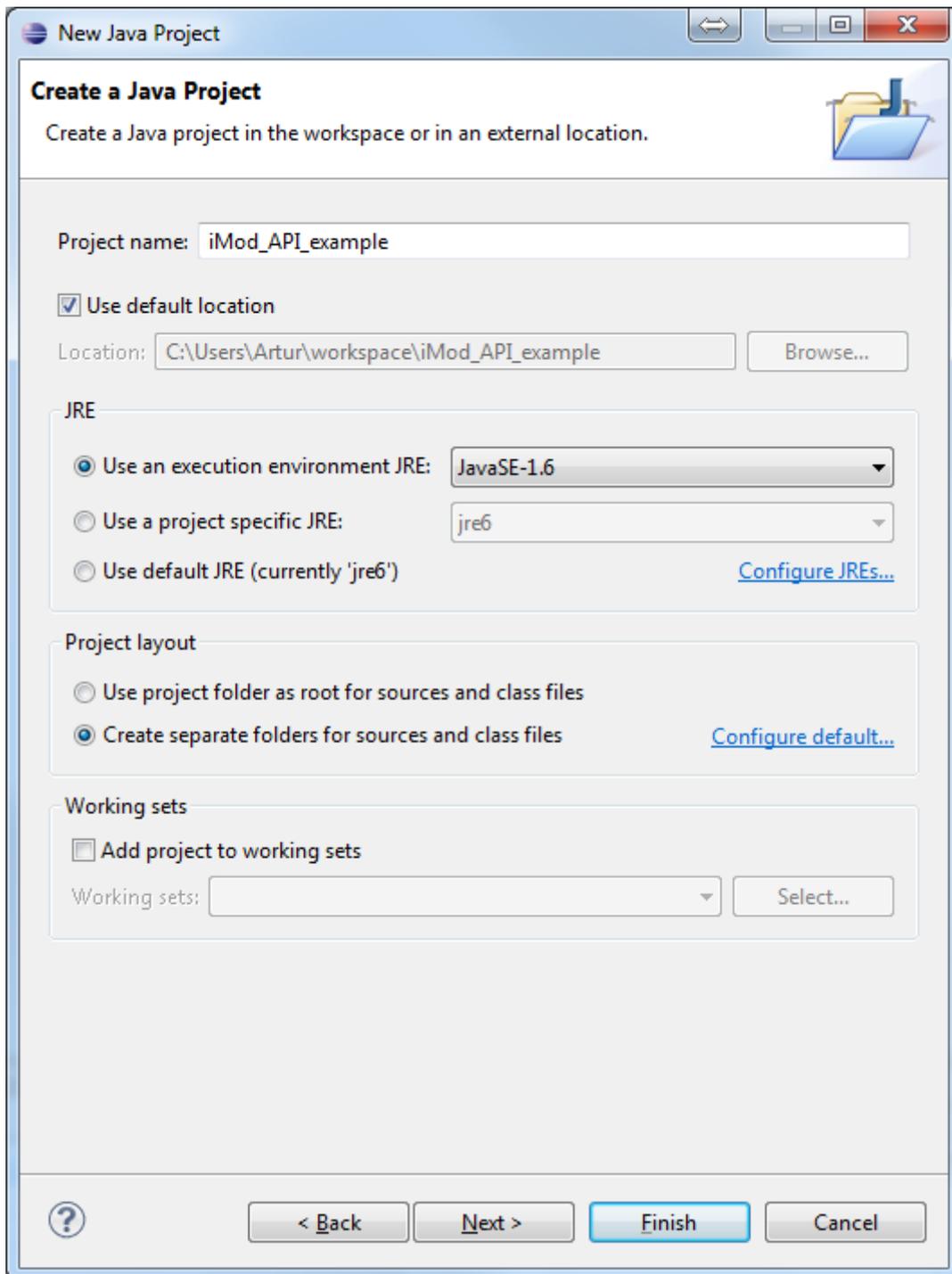
Channel implementation will need imodlib and commons-logging library.

Preparation of the project in Eclipse IDE

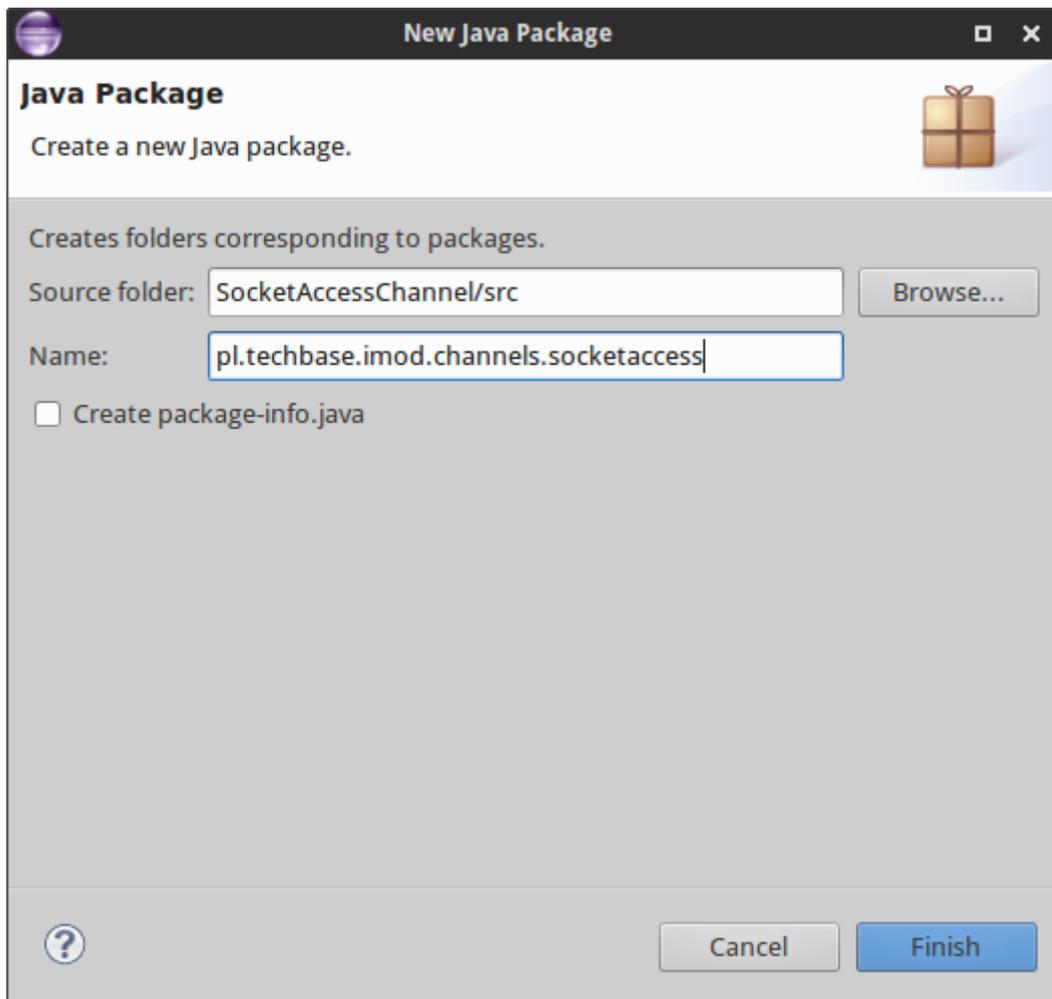
- 1) Open Eclipse and create new Project
- 2) Choose Java Project and click next



- 3) Configure project. Choose name (i.e. iModSDKMessageChannel) and select java version to 1.6 and press *Finish* button



4) Add a package that is compliant with SDK requirements (File → new → package):
pl.techbase.imod.channels.name In our case: pl.techbase.imod.channels.examplechan



5) In the main project directory, create the lib subdirectory. Next, copy imodlib.jar, commons-logging-1.1.1.jar and javolution-5.5.1.jar to it.

6) Return now to eclipse and refresh project (right click on project's name in Package Explorer and choose Refresh action or press F5 button. You will see our lib directory with two libraries. Add both of them to the classpath: right click on each of them and choose Build Path → Add to build path.

7) Add build.xml file to main project's directory. This file will be responsible for compilation and creation of the jar archive with our custom channel. Put following content in build.xml file:

```
<?xml version="1.0" encoding="iso-8859-2"?>
<project name="Example date plugin for iMod - create package" basedir="."
default="make_jar">
  <property name="techbase.name" value="Customer" />
  <property name="build.dir" location="builder/build/" />
  <property name="class.dir" location="bin/pl" />
  <target name="clean">
    <delete dir="${build.dir}" />
  </target>
  <!--
////////////////////////////////////prepare////////////////////////////////////
-->
  <target name="prepare" depends="clean">
```

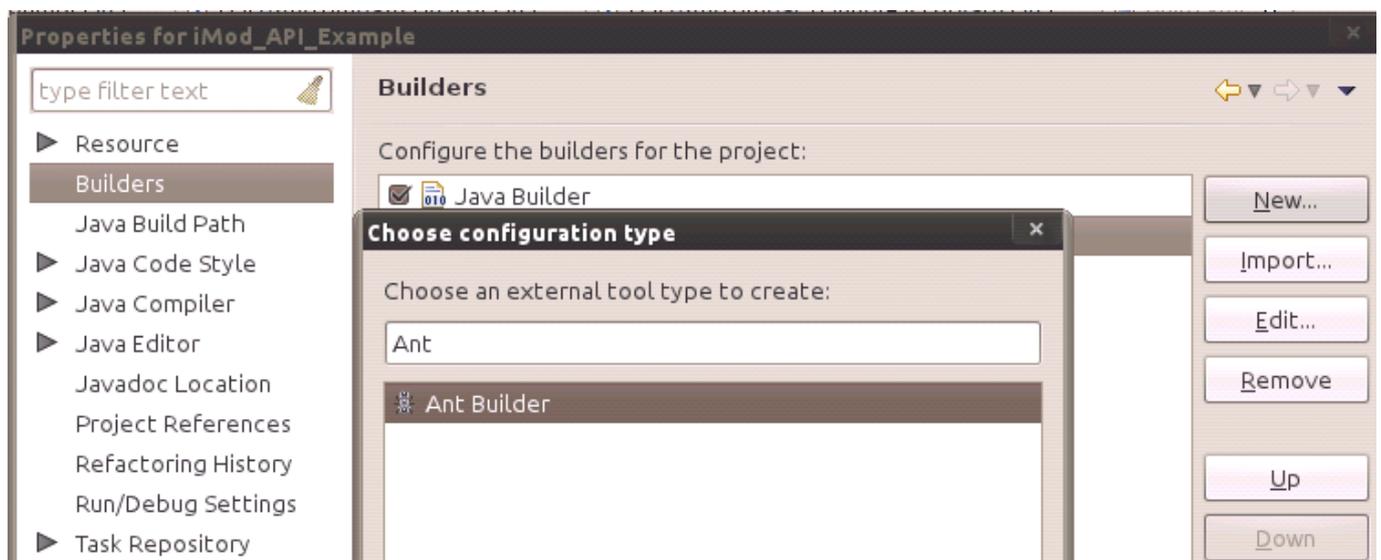
```

    <mkdir dir="${build.dir}" />
</target>
<!-- ////////////////////////////////////////////make_jar////////////////////////////////////////// -->
->
<target name="make_jar" depends="prepare">
    <jar destfile="${build.dir}/examplmchan.jar">
        <fileset dir="bin">
            <include name="**/*.class" />
        </fileset>
        <manifest>
            <attribute name="Built-By" value="${techbase.name}"/>
            <attribute name="Class-Path" value="imodlib.jar commons-logging-1.1.1.jar javolution-5.5.1.jar" />
        </manifest>
    </jar>
</target>
</project>

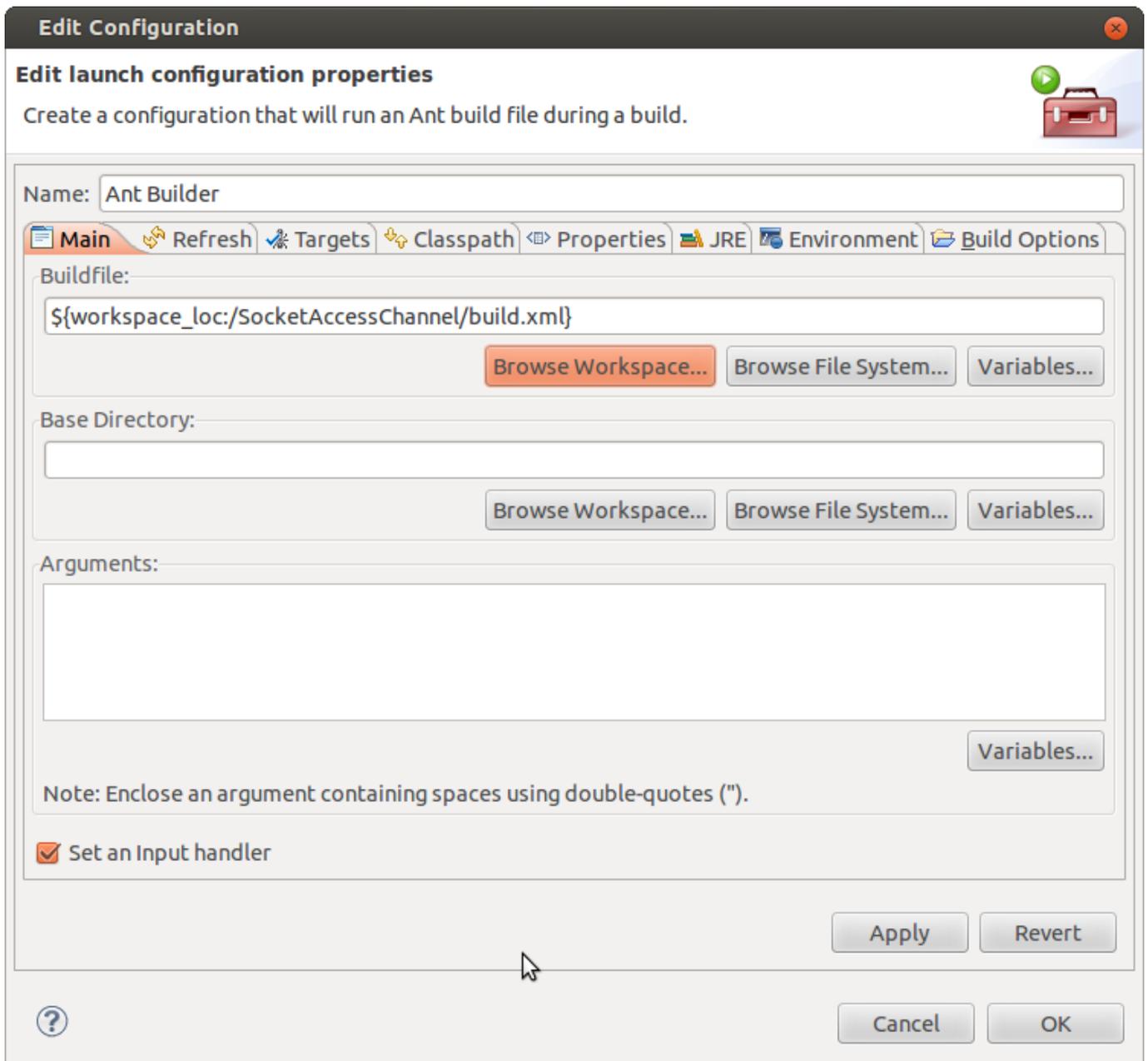
```

8) Configure ant builder:

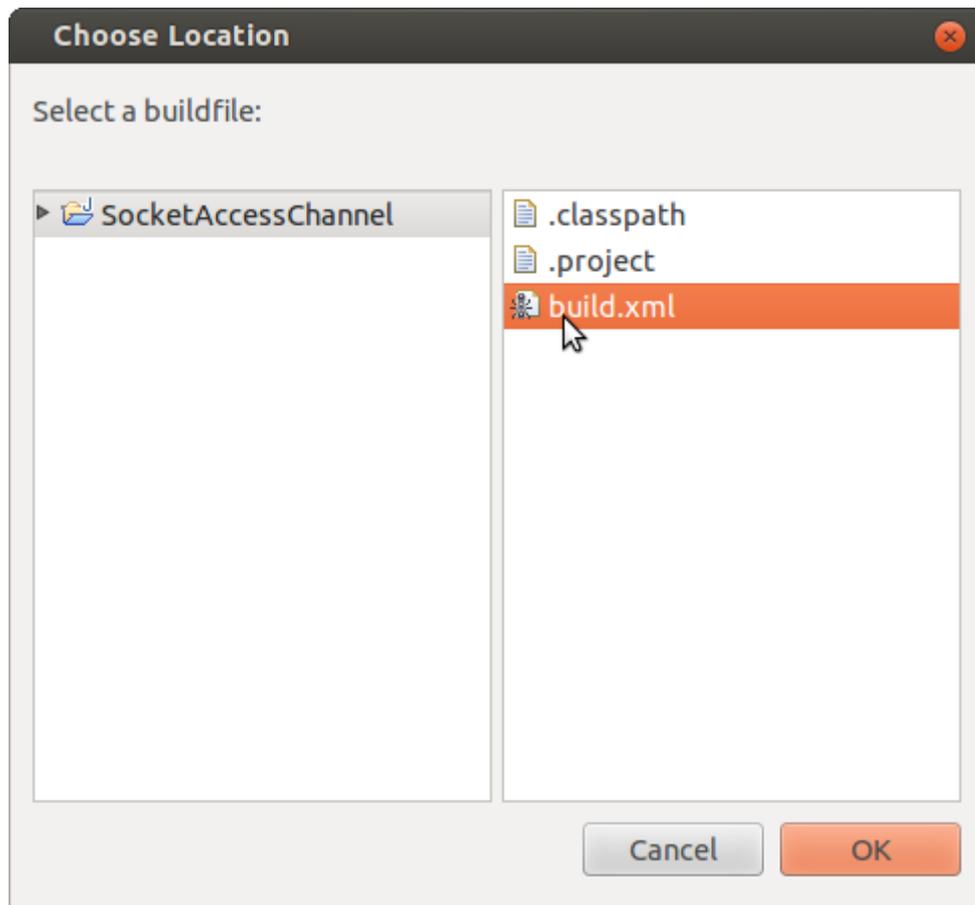
- Go to project's properties (right click on project and choose *Properties*)
- Choose *Builders* position in left panel
- Click *New...* button and choose *Ant Builder*



- In *Edit Configuration* window type name for our builder...



- ...and select build.xml file (click *Browse Workspace...* button in *Buildfile* section)



Code writing

1) Now we will implement all three necessary for every channel classes:

- EXAMPLECHANMActivator
- EXAMPLECHANMChannelFactorySrv
- EXAMPLECHANMChannelImpl

2) In EXAMPLECHANMActivator put code of our server:

```
package pl.techbase.imod.channels.exemplechan;

import pl.techbase.imod.imodlib.common.CommonActivator;
import pl.techbase.imod.imodlib.plugins.ISChannelFactory;

public class EXAMPLECHANMActivator implements CommonActivator {

    public EXAMPLECHANMActivator() {
        super();
    }

    @Override
    public ISChannelFactory getFactory() {
        return new EXAMPLECHANMChannelFactorySrv();
    }
}
```

```
}
```

3) In EXAMPLEMCHANMChannelFactorySrv put following code:

```
package pl.techbase.imod.channels.exemplemchan;

import java.util.ArrayList;
import java.util.List;
import pl.techbase.imod.imodlib.engine.IChannelProxy;
import pl.techbase.imod.imodlib.plugins.IChannel;
import pl.techbase.imod.imodlib.plugins.ISChannelFactory;

public class EXAMPLEMCHANMChannelFactorySrv implements ISChannelFactory {

    private List<IChannel> channelList = new ArrayList<IChannel>();

    @Override
    public IChannel newInstance(IChannelProxy iModEngine, boolean scanmode)
    {
        IChannel channel = new EXAMPLEMCHANMChannelImpl(iModEngine,
scanmode);
        if (channel != null)
            this.channelList.add(channel);
        return channel;
    }

    @Override
    public void close() {
        if (this.channelList.size() > 0) {
            for (IChannel channel : this.channelList)
                channel.close();
        }
    }
}
```

4) In put EXAMPLEMCHANMChannelImpl implementation of channel:

```
package pl.techbase.imod.channels.exemplemchan;

import java.io.FileOutputStream;
import java.io.PrintStream;
import java.util.Iterator;
import java.util.List;

import org.apache.commons.logging.Log;
import org.apache.commons.logging.LogFactory;

import pl.techbase.imod.imodlib.common.CommonMChannel;
```

```
import pl.techbase.imod.imodlib.common.IModException;
import pl.techbase.imod.imodlib.engine.Error;
import pl.techbase.imod.imodlib.engine.IChannelProxy;
import pl.techbase.imod.imodlib.engine.IData.Status;
import pl.techbase.imod.imodlib.engine.IEvent;
import pl.techbase.imod.imodlib.engine.IProperties;
import pl.techbase.imod.imodlib.plugins.IChannel;

public final class EXAMPLEMCHANMChannelImpl extends CommonMChannel
implements IChannel {

    // this object allow us write some messages to iMod logs
    private static Log log =
LogFactory.getLog(EXAMPLEMCHANMChannelImpl.class);
    // additional property value. This value will be put before message in
    // output files
    private String testproperty = "";

    // Initialize channel
    public EXAMPLEMCHANMChannelImpl(IChannelProxy iModEngine, boolean
scanmode) {
        // call parent's constructor
        super(iModEngine, scanmode);
        // set name of current channel
        this.setName(this.getChanName());
    }

    // this method allow configure our channel
    @Override
    protected boolean configure() throws IModException {
        // call base implementation's configure method. This will load
        // properties defined from all message channels, for example
recipients
        if (super.configure()) {
            // parse channel's parameters from MainConfig.xml
            Iterator<java.util.Properties> itr =
this.m_channel_proxy.getConfig().iterator();
            while (itr.hasNext()) {
                java.util.Properties p = itr.next();
                Iterator<Object> itrk = p.keySet().iterator();

                while (itrk.hasNext()) {
                    String key = (String) itrk.next();
                    String value = (String) p.get(key);

                    // here we can parse our channel properties from xml

                    // This construction (property.property_name) matches
                    // <property name="property_name" value="v" /> from xml

```

```

        if (key.equals("property.testproperty"))
            testproperty = value;
    }
}

// configuration succeed
return true;
} else
    // configuration failed
    return false;
}

// this method is called when event condition are met
@Override
protected Status transmit(IEvent e, Object value) {
    boolean res = true;

    // get message's value
    String message = e.getMessage().toString();

    log.debug("Saving message " + message + " to defined files");

    // Iterate through recipients from channel's definition
    if (this.m_recipients != null)
        for (String recipient : this.m_recipients) {
            res = saveToFile(recipient, message);
        }

    // Get recipient defined in the event
    String localRecipient = e.getLocalRecipient();
    if (localRecipient != null && !localRecipient.isEmpty()) {
        res = saveToFile(localRecipient, message);
    }

    if (res) // If everything is ok, return S_OK status
        return Status.S_OK;
    else
        // Some write failed. Return S_LAST_WRITE_FAILED status. iMod
will
        // try call transmit method again with the same data
        return Status.S_LAST_WRITE_FAILED;
}

// Method which saves message to file
private boolean saveToFile(String path, String txt) {
    try {
        FileOutputStream fos = new FileOutputStream(path, true);
        PrintStream ps = new PrintStream(fos);
        ps.println(testproperty + " " + txt);
    }
}

```

```

        ps.close();
        fos.close();
        log.debug("Message " + txt + " has been saved to file " +
path);

        // Data has been saved properly, return true
        return true;
    } catch (Exception e) {
        log.error("Print line to file failed: " + e.getMessage());
        // Data saving failed, return false
        return false;
    }
}

// Following methods can have no particular implementation
@Override
public Object getParameter(String id) {
    return null;
}

@Override
public List<Error> getStatus() {
    return null;
}

@Override
public void setConfig(IProperties config) {
}
}

```

Building

1) To build our Example Message Channel channel right click on build.xml file and choose *Run As* and *Ant Build* option (without dots). If some failure message appear, try build again. Our custom channel library will appear in builder/build directory.

Deploying

x500/x500cm3

After concluding correct implementation of the custom message channel and creation of the JAR archive, copy jar archive to directory /home/imod/iModTiger-dev/libs/ on device and restart iMod.

9000/x1000

After concluding correct implementation of the custom message channel and creation of the JAR archive, modify the iModEngine manifest so that the application kernel can see the class of the new channel:

1) Acquire imodTiger.jar file from device (this library is in the main directory of iMod application in npe device).

2) After downloading the file, unpack the manifest file using the command:

```
jar xf imodTiger.jar META-INF/MANIFEST.MF
```

3) Add an entry to the Class-Path line with the target location of the socketaccess.jar file:

```
Manifest-Version: 1.0
Ant-Version: Apache Ant 1.8.2
Built-By: TECHBASE Group sp. z o.o.
Class-Path: ../jar/libs/snmp4j-1.11.1.jar ../jar/libs/sqlitejdbc-v056.
jar ../jar/libs/commons-logging-1.1.1.jar ../jar/libs/log4j-1.2.16.ja
r ../jar/protocols/imodlib.jar ../jar/protocols/modbus.jar ../jar/pro
tocols/mbus.jar ../jar/protocols/ecl300.jar ../jar/protocols/onewire.
jar ../jar/protocols/archivist.jar ../jar/protocols/snmp.jar ../jar/p
rotocols/rfid.jar ../jar/protocols/mssql.jar ../jar/libs/mail.jar ../
jar/libs/smslib.jar ../jar/libs/jowfsclient.jar ../jar/libs/slf4j-api
-1.5.8.jar ../jar/libs/slf4j-log4j12-1.5.8.jar ../jar/libs/protobuf-j
ava-2.4.1-lite.jar ../jar/libs/zmq.jar ../jar/libs/cron4j-2.2.5.jar .
../jar/libs/postgresql-8.4-703.jdbc4.jar ../jar/libs/commons-io-2.3.ja
r ../jar/libs/xmlrpc-server-3.1.3.jar ../jar/libs/xmlrpc-common-3.1.3
.jar ../jar/libs/xmlrpc-client-3.1.3.jar ../jar/libs/ws-commons-util-
1.0.2.jar ../jar/libs/jedis-2.0.0.jar ../jar/libs/jtds-1.2.8.jar ../j
ar/protocols/Console_parser.jar ../jar/libs/mysql-connector-java-5.1.
26-bin.jar ../jar/protocols/exemplmchan.jar
Created-By: 1.6.0_32-b32 (Sun Microsystems Inc.)
Main-Class: pl.techbase.imod.Main
```

4) Next, using the following command, update the manifest in the imodTiger file:

```
jar umf META-INF/MANIFEST.MF imodTiger.jar
```

After finishing the update, the new imodTiger.jar file is to be saved to the main iMod application directory on the NPE device. Save the newly created exemplmchan.jar file implementing our custom message channel to the directory indicated in the Manifest (jar/protocols).

Testing custom channel

Example configuration:

```
<?xml version="1.0"?>
<imod version="1.0.0">
  <group name="Channels">
    <message-channel name="ExampleMChannel">
      <protocol name="EXAMPLEMCHAN"/>
      <recipient>"/root/testfile1.txt"</recipient>
      <recipient>"/root/testfile2.txt"</recipient>
      <property name="testproperty" value="AA" />
    </message-channel>
  </group>
</imod>
```

```

    <access-channel name="ModbusTCP">
      <protocol name="MODBUS"/>
      <port>"ET-502-TCP"</port>
    </access-channel>
  </group>
  <group name="Messages">
    <message id="message"><![CDATA["Value of parameter REG_NAME[THIS]:
REG_VALUE[THIS]"]]></message>
  </group>
  <group name="Parameters">
    <parameter>
      <id>100</id>
      <access-channel channel-name="ModbusTCP" parameter-id="100"/>
      <event type="OnChange">
        <message-channel channel-name="ExampleMChannel" parameter-
id="/root/testfile3.txt"/>
        <message-id>"message"</message-id>
      </event>
    </parameter>
  </group>
</iMod>

```

Upload example configuration to file `/mnt/mtd/iMod/config/MainConfig.xml` on the device.

To run iMod with our changes execute `imod` command with argument `start`, `trace` or `debug` on npe device:

```
imod start
```

Now iMod should save message defined in message definition called `message` with prefix from our custom channel's property `testproperty` to files defined in recipients. Configuration defines three recipients. Two of them are in the channel's definition in `<recipient>` nodes. Third is set in `parameter-id` in event's definition.

To trigger this event change value of `100` parameter with `modmas` application:

```
modmas write:100:12
```

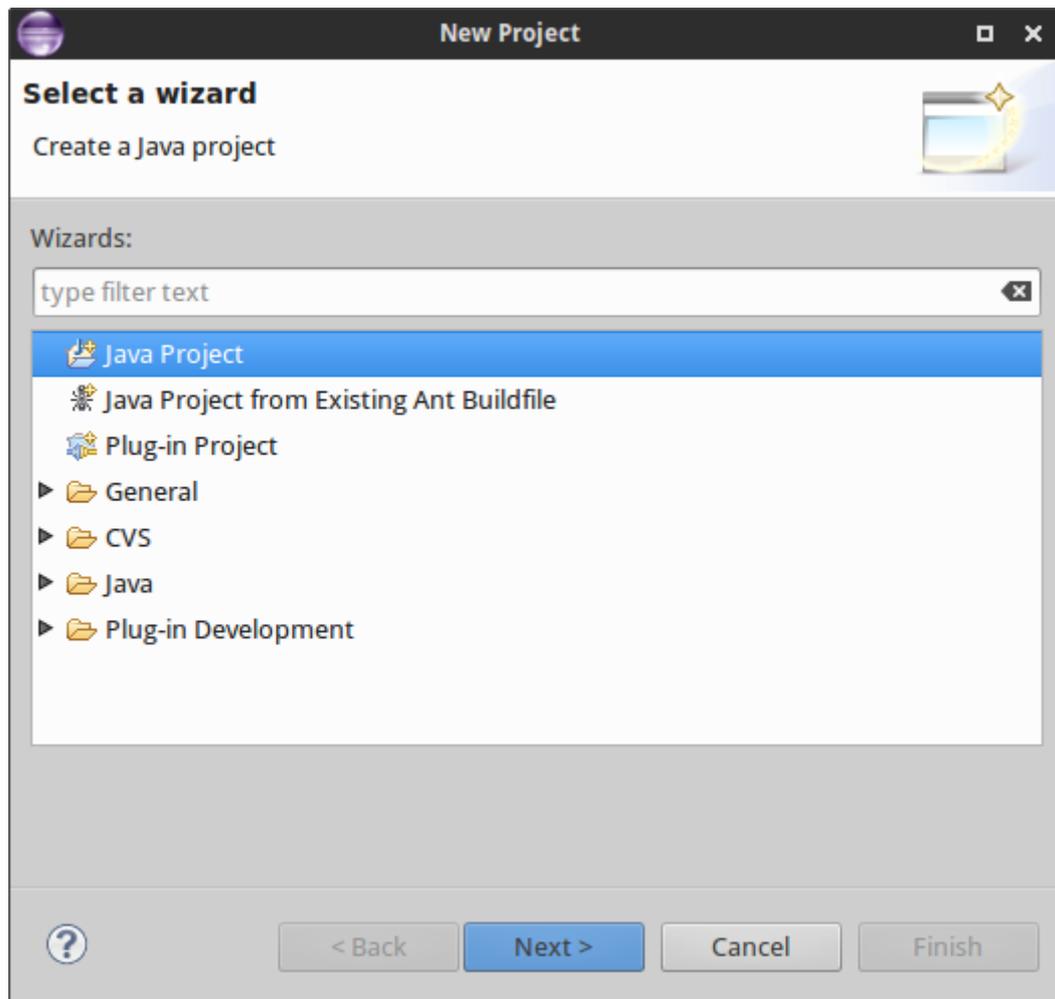
Custom access-channel

In this instruction we will add easy access channel to `imod` using `iModSDK`. We will use `Sockets` from `Java` to get or set parameters in `iMod`. Our channel will be named `SOCKETACCESS`. We will also need client for our channel which will be simple command-line program.

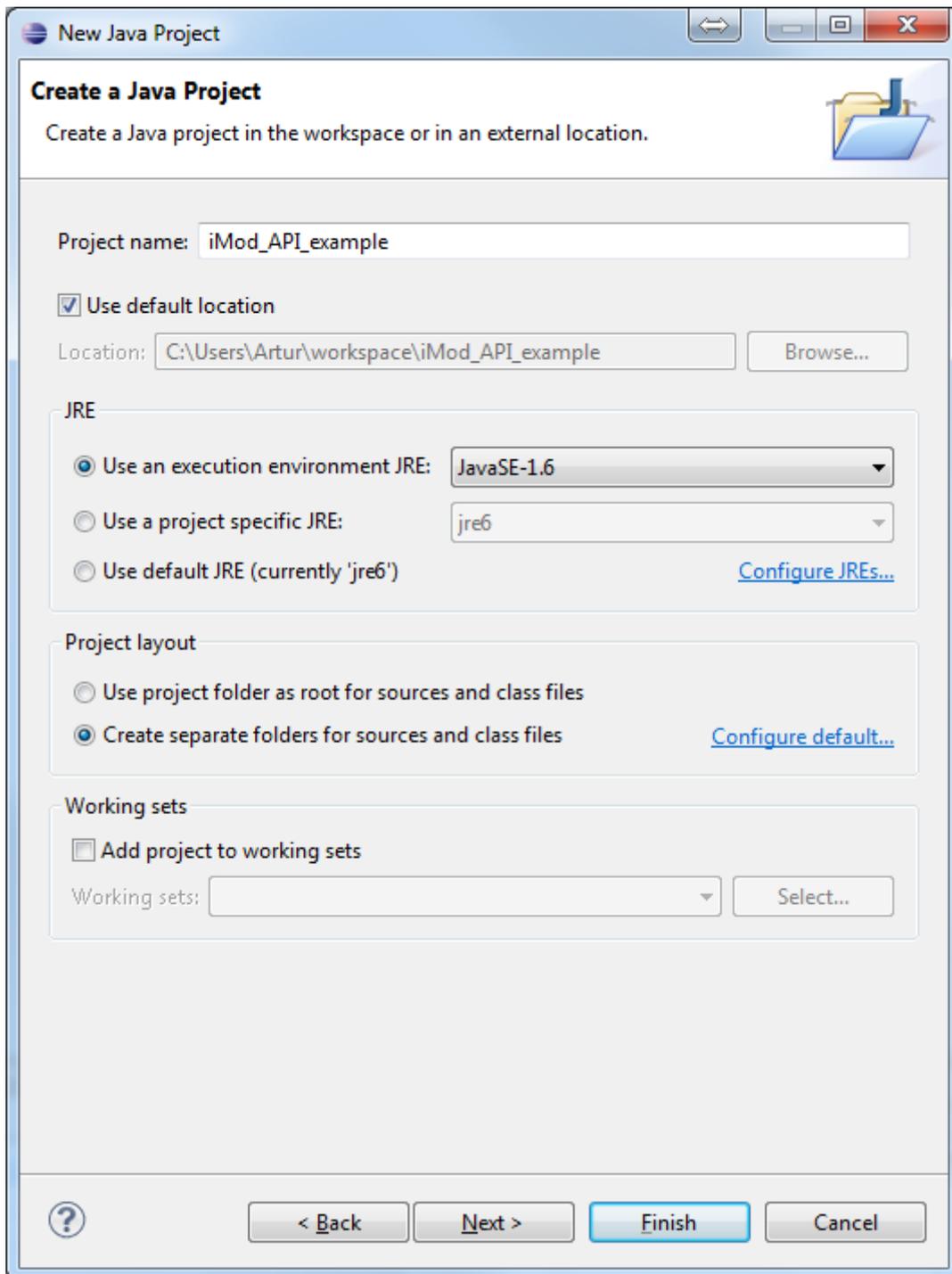
Server

Preparation of the project in Eclipse IDE

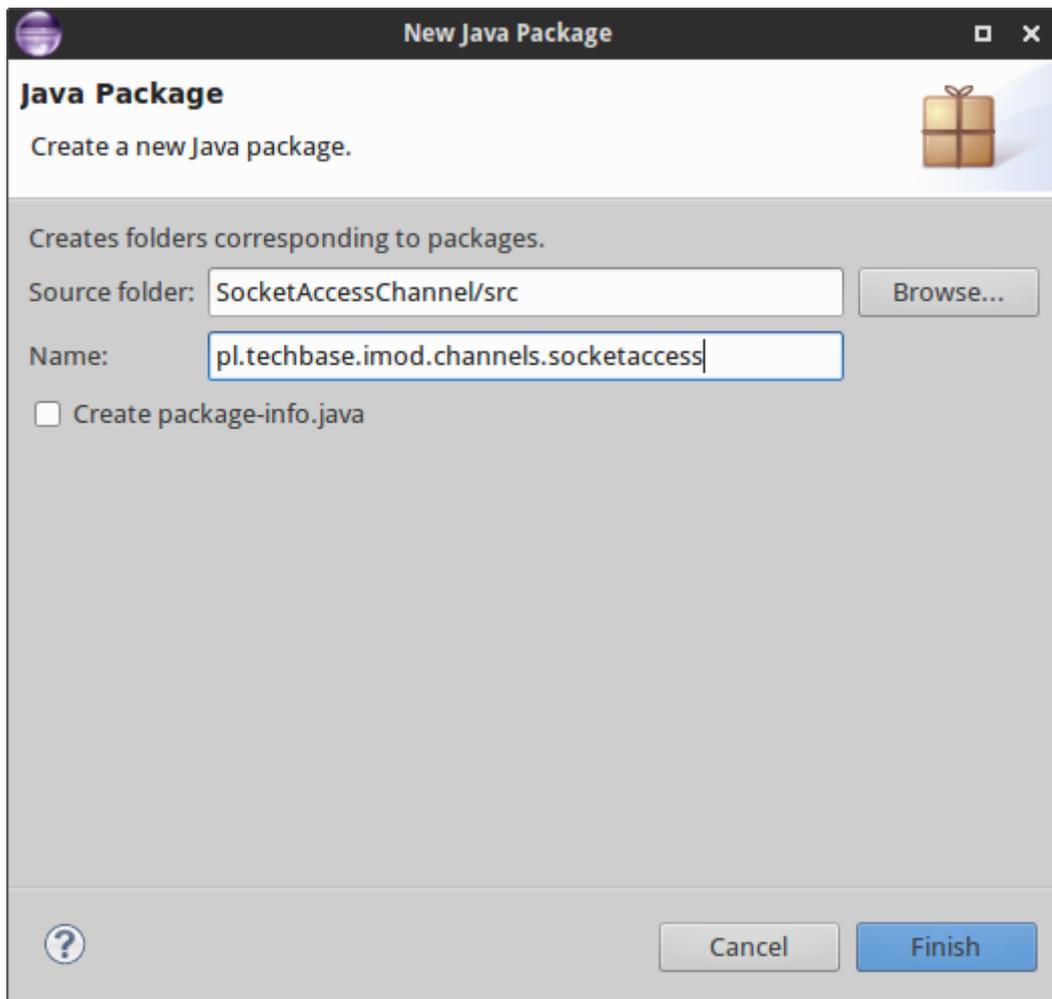
- 1) Open Eclipse and create new Project
- 2) Choose Java Project and click next



3) Configure project. Choose name (i.e. socketaccess) and select java version to 1.6 and press *Finish* button



4) Add a package that is compliant with SDK requirements (File → new → package):
`pl.techbase.imod.channels.name` In our case: `pl.techbase.imod.channels.socketaccess`



5) In the main project directory, create the lib subdirectory. Next, copy imodlib.jar to it. File imodlib.jar can be found in /jar/protocols in the iMod application installation directory on device.



The main installation directory of the iMod application can be checked using the command: `getenv IMOD_VERSION`

6) In our example we will use Log4j library to write some debug information to console. So we need also commons-logging-1.1.1.jar file which can be found in /jar/libs directory in the iMod application installation location. Copy this file to our lib directory.

7) Return now to eclipse and refresh project (right click on project's name in Package Explorer and choose Refresh action or press F5 button. You will see our lib directory with two libraries. Add both of them to the classpath: right click on each of them and choose Build Path → Add to build path.

8) Add build.xml file to main project's directory. This file will be responsible for compilation and creation of the jar archive with our custom channel. Put following content in build.xml file:

```
<?xml version="1.0" encoding="iso-8859-2"?>
<project name="Example date plugin for iMod - create package" basedir="."
```

```

        default="make_jar">
<property name="techbase.name" value="Customer" />
<property name="build.dir" location="builder/build/" />
<property name="class.dir" location="bin/pl" />
<target name="clean">
    <delete dir="${build.dir}"/>
</target>
<!--
////////////////////////////////////prepare////////////////////////////////////
-->
<target name="prepare" depends="clean">
    <mkdir dir="${build.dir}" />
</target>
<!-- //////////////////////////////////make_jar//////////////////////////////////// -
->
<target name="make_jar" depends="prepare">
    <jar destfile="${build.dir}/socketaccess.jar">
        <fileset dir="bin">
            <include name="**/*.class"/>
        </fileset>
        <manifest>
            <attribute name="Built-By" value="${techbase.name}"/>
            <attribute name="Class-Path" value="imodlib.jar commons-
logging-1.1.1.jar" />
        </manifest>
    </jar>
</target>
</project>

```

It is important to modify this two parameters: destfile and main.class:

- destfile: In this parameter we can set name for output jar file with custom channel:

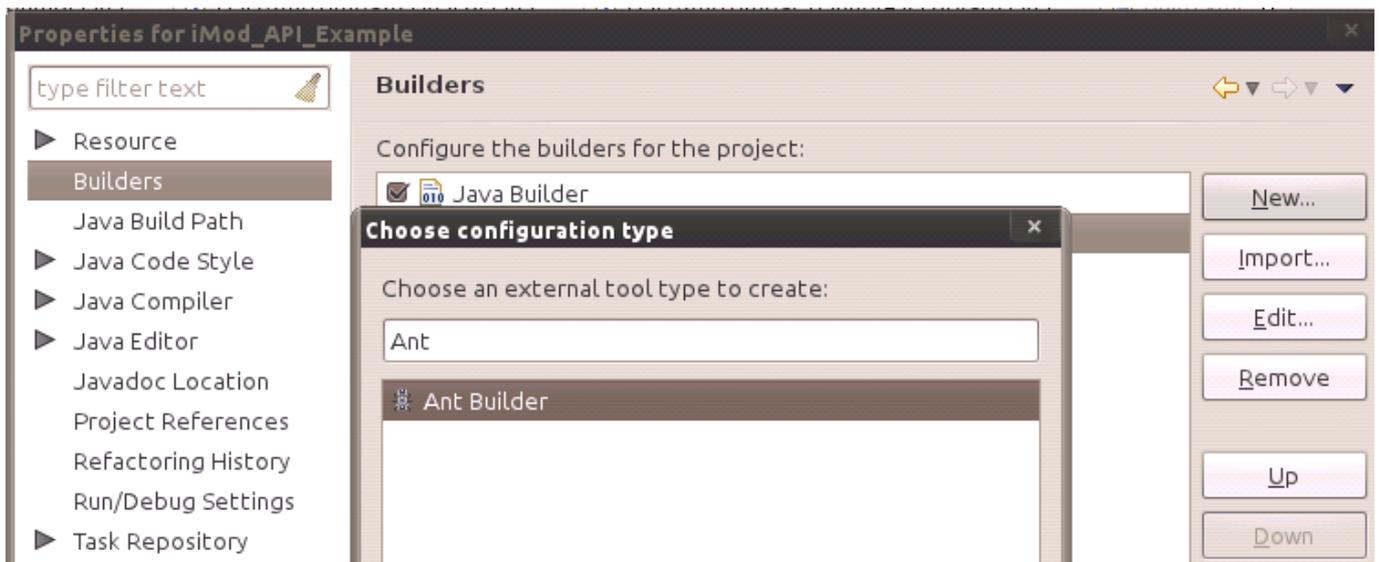
```
<jar destfile="${build.dir}/jar_name.jar">
```

- main.class: In this parameter we set main class of our channel:

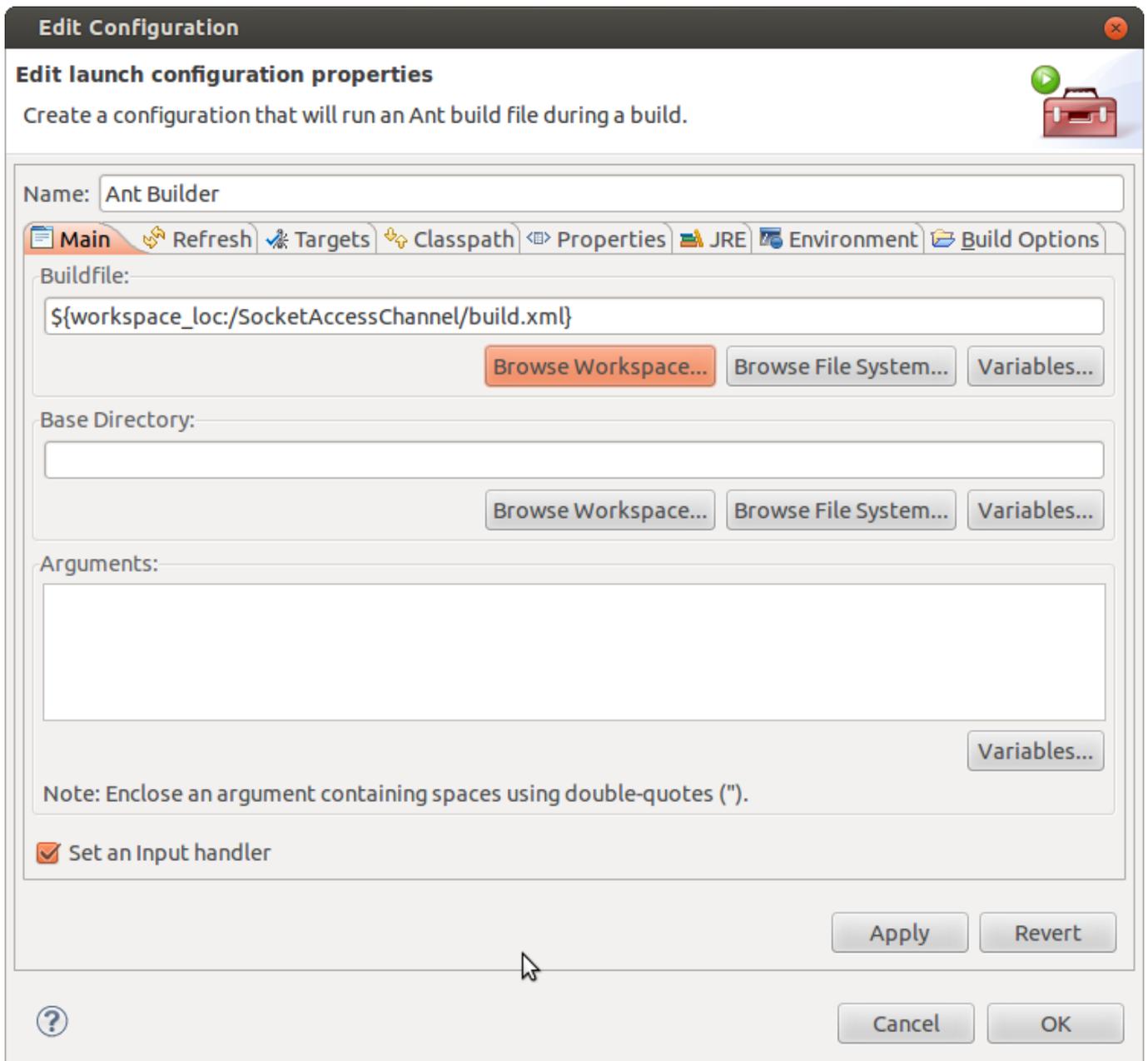
```
<property name="main.class" value="" />
```

9) Configure ant builder:

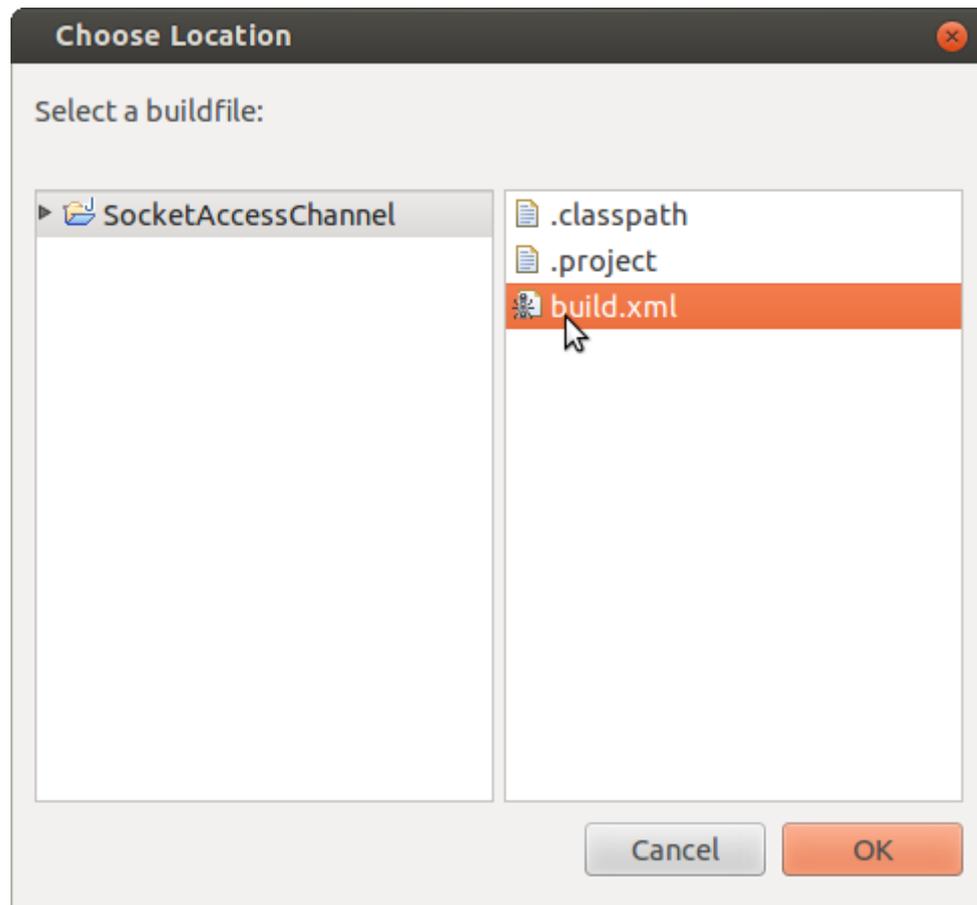
- Go to project's properties (right click on project and choose *Properties*)
- Choose *Builders* position in left panel
- Click *New...* button and choose *Ant Builder*



- In *Edit Configuration* window type name for our builder...



- ...and select build.xml file (click *Browse Workspace...* button in *Buildfile* section)



Code writing

1) Now we will add code of SocketAccess channel.

Create four java classes in *pl.techbase.imod.channels.socketaccess* package:

- SOCKETACCESSAActivator
- SOCKETACCESSAChannelFactorySrv
- SOCKETACCESSAChannelImpl
- SocketAccessServer

First three files are obligatory for each custom channel. SocketAccessServer.java will contain code of our socket server. Following codes contain comments that explain how channel works.



It is important to set custom channel's classes names using CAPITAL LETTERS. Otherwise kernel won't find our custom channel

SocketAccess channel will have two parameters:

- port - defines on which port socket server will listen. Value: integer
- enable_write - this property defines if client will be able to change parameters in imod

This snippet shows how to define SocketAccess channel in MainConfig.xml:

```
<access-channel name="SocketAccess_1">
  <protocol name="SocketAccess"/>
  <port>1234</port>
  <property name="enable_write" value="true" />
</access-channel>
```

2) In SocketAccessServer put code of our server:

```
package pl.techbase.imod.channels.socketaccess;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.PrintStream;
import java.net.ServerSocket;
import java.net.Socket;
import java.net.SocketException;

import org.apache.commons.logging.Log;
import org.apache.commons.logging.LogFactory;

import pl.techbase.imod.imodlib.engine.IChannelProxy;
import pl.techbase.imod.imodlib.engine.IData;

//server's class must implement Runnable interface to work as a separate
//thread
public class SocketAccessServer implements Runnable {
    // this object allow us write some messages to iMod logs
    private static Log log = LogFactory.getLog("SocketAccessServer");
    // this flag will show us if we should stop server's thread
    private boolean running = false;
    // port number which server will listen to
    private int port;
    // this object allow us read and write parameters from iMod
    private IChannelProxy proxy;
    // thread object. Our server must work in separate thread
    private Thread thread;
    // socket for server. We will use it to wait for client
    private ServerSocket serverSocket = null;
    // flag which indicates if we can change value for property
    private boolean isWriteEnabled;

    // Initialize server's class with port number, proxy object and allow
    // write
    // flag
    public SocketAccessServer(int port, IChannelProxy proxy, boolean
enableWrite) {
        this.port = port;
    }
}
```

```

        this.proxy = proxy;
        this.isWriteEnabled = enableWrite;
    }

    public void run() {
        // this prevent this method to run outside of new thread. start()
method
        // sets this flag to true and then creates new thread with this
method
        if (!running)
            return;
        try {
            // initialize server socket with specified port
            serverSocket = new ServerSocket(port);
            while (true) {
                log.debug("waiting for request from client");
                // if imod interrupts our channel before we start listening
this
                // code will stop server
                if (!running) {
                    serverSocket.close();
                    log.debug("SocketAccessServer stopped");
                    return;
                }
                // start blocking listening
                Socket client = serverSocket.accept();
                // if somebody connect we serve client
                // there can be only one client at the same time
                serveClient(client);
                if (!running) {
                    serverSocket.close();
                    log.debug("SocketAccessServer stopped");
                    return;
                }
            }
        }

        } catch (SocketException e) {
            if (!running) {
                log.debug("SocketAccessServer stopped");
            } else {
                log.error("Socket error");
                running = false;
            }
        }
        } catch (IOException e) {
            log.error("SocketAccess server run error");
            running = false;
            log.error(e.getMessage());
        }
        } catch (Exception e) {
            log.error(e.getStackTrace());
        }
    }
}

```

```

        running = false;
    }
}

// this method contains code to serve the client
private void serveClient(Socket client) throws IOException {
    log.debug("request accepted");
    // get reader for input stream from client
    BufferedReader br = new BufferedReader(new InputStreamReader(
        client.getInputStream()));
    // create PrintStream. This object will be used to send some data
to
    // client
    PrintStream ps = new PrintStream(client.getOutputStream());
    while (true) {
        if (!running) {
            log.debug("stopping work");
            ps.println("ServerAccess is stopping now");
            client.close();
            break;
        }
        String st = br.readLine();
        log.debug(st);
        if (st == null || st.equals("exit") == true) {
            log.debug("client has ended connection...");
            client.close();
            break;
        } else {
            // parse input
            // we are waiting for messages in following formats:
            // read:param_number
            // write:param_number:new_value
            // Split message by colon
            // parts table will contain parts of proper message
            String[] parts = st.split(":");

            if (parts.length < 2 || parts.length > 3) {
                ps.println("Bad input: " + st);
            } else {
                String actionName = parts[0]; // read or write
                String parameterId = parts[1];
                // null if message doesn't contain value:
                String newValue = parts.length == 3 ? parts[2] : null;
                if (actionName.equals("read")) {
                    // read parameter from iMod:
                    IData param = proxy.getParameter(parameterId);
                    // if parameter doesn't exist send appropriate
message
                    // to client

```

```

        if (param == null) {
            ps.println("Parameter " + parameterId
                + " doesn't exist");
        } else {
            // if message exists get value from it
            ps.println("Value for parameter " + parts[1] +
                + param.getValue());
        }
    } else if (actionName.equals("write") && newValue !=
null) {
        if (!isWriteEnabled) {
            ps.println("Write not allowed");
        } else {
            try {
                // set new value for parameter if new value
is
                // integer
                proxy.setParameter(parameterId,
                    Integer.parseInt(newValue));
                ps.println("Parameter updated");
            } catch (NumberFormatException e) {
                ps.println("Bad value for write: " +
newValue
                    + ". Should be an integer");
            }
        }
    }
}
}
}
}
}
}
}
}
}

// start server
public void start() {
    // don't start server if it is already running
    if (running)
        return;
    running = true;
    // create new thread
    thread = new Thread(this);
    // and start it
    thread.start();
}

// stop server
// server will stop when it check running flag or immediately if server
is
// in listening mode (is waiting for client in accept() function)

```

```
public void stop() {
    if (running) {
        running = false;
        if (serverSocket != null)
            try {
                serverSocket.close();
            } catch (IOException e) {
            }
        }
    }
}
```

3) In SOCKETACCESSAActivator put following code:

```
package pl.techbase.imod.channels.socketaccess;

import pl.techbase.imod.imodlib.common.CommonActivator;
import pl.techbase.imod.imodlib.plugins.ISChannelFactory;

public class SOCKETACCESSAActivator implements CommonActivator {

    @Override
    public ISChannelFactory getFactory() {
        return new SOCKETACCESSChannelFactorySrv();
    }
}
```

4) In SOCKETACCESSChannelFactorySrv put following code:

```
package pl.techbase.imod.channels.socketaccess;

import java.util.ArrayList;
import java.util.List;

import pl.techbase.imod.imodlib.engine.IChannelProxy;
import pl.techbase.imod.imodlib.plugins.IChannel;
import pl.techbase.imod.imodlib.plugins.ISChannelFactory;

public class SOCKETACCESSChannelFactorySrv implements ISChannelFactory {
    private List<IChannel> channelList = new ArrayList<IChannel>();
    @Override
    public void close() {
        if (this.channelList.size() > 0) {
            for (IChannel channel : this.channelList)
                channel.close();
        }
    }
}
```

```

@Override
public IChannel newInstance(IChannelProxy iModEngine, boolean scanmode)
{
    IChannel channel = new SOCKETACCESSChannelImpl(iModEngine,
scanmode);
    if (channel != null)
        this.channelList.add(channel);
    return channel;
}
}

```

5) In SOCKETACCESSChannelImpl put implementation of SocketAccess channel:

```

package pl.techbase.imod.channels.socketaccess;

import java.util.Iterator;
import java.util.List;

import org.apache.commons.logging.Log;
import org.apache.commons.logging.LogFactory;

import pl.techbase.imod.imodlib.common.CommonChannel;
import pl.techbase.imod.imodlib.common.IModException;
import pl.techbase.imod.imodlib.engine.Error;
import pl.techbase.imod.imodlib.engine.IChannelProxy;
import pl.techbase.imod.imodlib.engine.IData.Status;
import pl.techbase.imod.imodlib.engine.IProperties;
import pl.techbase.imod.imodlib.plugins.IChannel;

public class SOCKETACCESSChannelImpl extends CommonChannel implements
IChannel {
    // this object allow us write some messages to iMod logs
    private static Log log =
LogFactory.getLog(SOCKETACCESSChannelImpl.class);
    // object which will be used in synchronized blocks
    private Object m_lock = new Object();
    // flag which indicate if this channel is enabled
    private volatile boolean m_enabled = true;
    // port with default value
    private int port = 1234;
    // enable write flag which is set with default value
    private boolean isWriteEnabled = true;
    // our socket server object
    private SocketAccessServer serv;

    // initialize channel
    public SOCKETACCESSChannelImpl(IChannelProxy iModEngine, boolean
scanmode) {

```

```

    super(iModEngine, scanmode);
    this.setName(this.getChanName());
    log.info("Starting SocketAccess Channel ...");
    this.start();
}

// this method allow configure our channel
protected boolean configure() throws IModException {
    if (super.configure()) {
        Iterator<java.util.Properties> itr = this.m_channel_proxy
            .getConfig().iterator();
        // parse channel's parameters from MainConfig.xml
        while (itr.hasNext()) {
            java.util.Properties p = itr.next();
            Iterator<Object> itrk = p.keySet().iterator();
            while (itrk.hasNext()) {
                String key = (String) itrk.next();
                String value = (String) p.get(key);
                // here we can parse our channel properties from xml
                if (key.equals("port")) {
                    try {
                        port = Integer.parseInt(value);
                        log.debug("Port set to " + port);
                    } catch (NumberFormatException e) {
                        log.warn("Error while parsing port number.
Defaulting to "
                                + port);
                    }
                }
                // This construction (property.property_name) matches
                // <property name="property_name" value="v" /> in xml
                if (key.equals("property.enable_write")) {
                    isWriteEnabled = Boolean.parseBoolean(value);
                    log.debug("Write enabled set to " +
isWriteEnabled);
                }
            }
        }
        // initialize our socket server object
        serv = new SocketAccessServer(port, this.m_channel_proxy,
            this.isWriteEnabled);
        return true;
    } else {
        return false;
    }
}

@Override

```

```
public Object getParameter(String arg0) {
    return null;
}

@Override
public List<Error> getStatus() {
    return null;
}

@Override
public void setConfig(IProperties arg0) {
}

@Override
public Status setParameter(String arg0, Object arg1) {
    return Status.S_OK;
}

@Override
// this method is used by iMod to disable this channel
public void disable() {
    synchronized (m_lock) {
        m_enabled = false;
    }
}

@Override
// this method is used by iMod to enable this channel
public void enable() {
    synchronized (m_lock) {
        m_enabled = true;
        m_lock.notify();
    }
}

@Override
// this method is called by iMod in order to run channel
public void work() throws InterruptedException, IModException {
    while (this.m_channel_proxy.getConfig() == null) {
        log.warn("SocketAccess Channel not yet configured");
        Thread.sleep(5000);
    }

    this.configure();
    serv.start();

    log.debug("Starting the SocketAccess channel: " + this.getName());
    while (!this.m_stop) {
```



```
./jar/libs/postgresql-8.4-703.jdbc4.jar ./jar/libs/commons-io-2.3.jar
./jar/libs/xmlrpc-server-3.1.3.jar ./jar/libs/xmlrpc-common-3.1.3
.jar ./jar/libs/xmlrpc-client-3.1.3.jar ./jar/libs/ws-commons-util-
1.0.2.jar ./jar/libs/jedis-2.0.0.jar ./jar/libs/jtds-1.2.8.jar ./j
ar/protocols/Console_parser.jar ./jar/libs/mysql-connector-java-5.1.
26-bin.jar ./jar/protocols/socketaccess.jar
Created-By: 1.6.0_32-b32 (Sun Microsystems Inc.)
Main-Class: pl.techbase.imod.Main
```

4) Next, using the following command, update the manifest in the imodTiger file:

```
jar umf META-INF/MANIFEST.MF imodTiger.jar
```

After finishing the update, the new imodTiger.jar file is to be saved to the main iMod application directory on the NPE device. Save the newly created socketaccess.jar file implementing the SocketAccess channel to the directory indicated in the Manifest (jar/protocols). After saving files, the parameters defined for the new channel can be added to the MainConfig.xml file for the purpose of checking the correctness of operation.

Testing custom channel

Example configuration: Turn on/off USER_LED using SocketAccess channel:

```
<imod version="1.0.0">
  <group name="Channels' definitions">
    <access-channel name="SocketAccess_1">
      <protocol name="SocketAccess"/>
      <port>1234</port>
      <property name="enable_write" value="true" />
    </access-channel>

    <source-channel name="NPE_io">
      <protocol name="HARDWARE"/>
    </source-channel>
  </group>
  <group name="Parameters">
    <parameter>
      <id>100</id>
      <description>"USR_LED"</description>
      <source-channel channel-name="NPE_io" parameter-id="USER_LED"/>
      <access-channel channel-name="SocketAccess_1"
parameter-id="100"/>
    </parameter>
  </group>
</imod>
```

Upload example configuration to file `/mnt/mtd/iMod/config/MainConfig.xml` on the device.

To run iMod with our changes execute `imod` command with argument `start`, `trace` or `debug` on npe

device:

```
imod start
```

Client

Creation of client

- 1) Create new java project
- 2) Add new package and name it for example socketaccessclient
- 3) Create class Main with *public static void main(String args[])* function.
- 4) Put following code:

```
package socketaccessclient;

import java.io.BufferedReader;
import java.io.InputStreamReader;
import java.io.PrintStream;
import java.net.Socket;

class SocketAccessClient {

    public static void main(String args[]) {
        //Initialize with default ip and port values
        String ip = "127.0.0.1";
        int port = 1234;
        //get custom ip and port from arguments
        if (args.length == 2) {
            ip = args[0];
            port = Integer.parseInt(args[1]);
        } else {
            System.out.println("To set custom ip and port give in first
argument ip and in second port");
        }
        System.out.println("IP set to " + ip + "\nPort set to " + port);
        System.out.println("sending request to server....");
        try {
            //create client socket and try connect to server
            //if connection fails IOException will be thrown
            Socket client = new Socket(ip, port);
            System.out.println("successfully connected");
            //create reader form standard input
            BufferedReader consoleReader = new BufferedReader(
                new InputStreamReader(System.in));
            //create object which will be used to send messages to server
            PrintStream clientSender = new
PrintStream(client.getOutputStream());
```

```
//create object to receive server's responses
BufferedReader serverResponseReader = new BufferedReader(
    new InputStreamReader(client.getInputStream()));
System.out.println("input the data you want to send to echo
server: ");
while (true) {
    System.out.print("Client: ");
    //read line from console
    String clientMessage = consoleReader.readLine();
    //send message to server
    clientSender.println(clientMessage);
    if (clientMessage.equals("exit")) {
        client.close();
        break;
    }
    //get response from server
    //if it is not first client, code will hang on this line
    //waiting for server's response which will never come
    String serverResponseMessage =
serverResponseReader.readLine();
    System.out.print("Server: ");
    System.out.println(serverResponseMessage);
}
} catch (Exception e) {
    e.printStackTrace();
}
}
```

5) Run project or export it and run from console with following command:

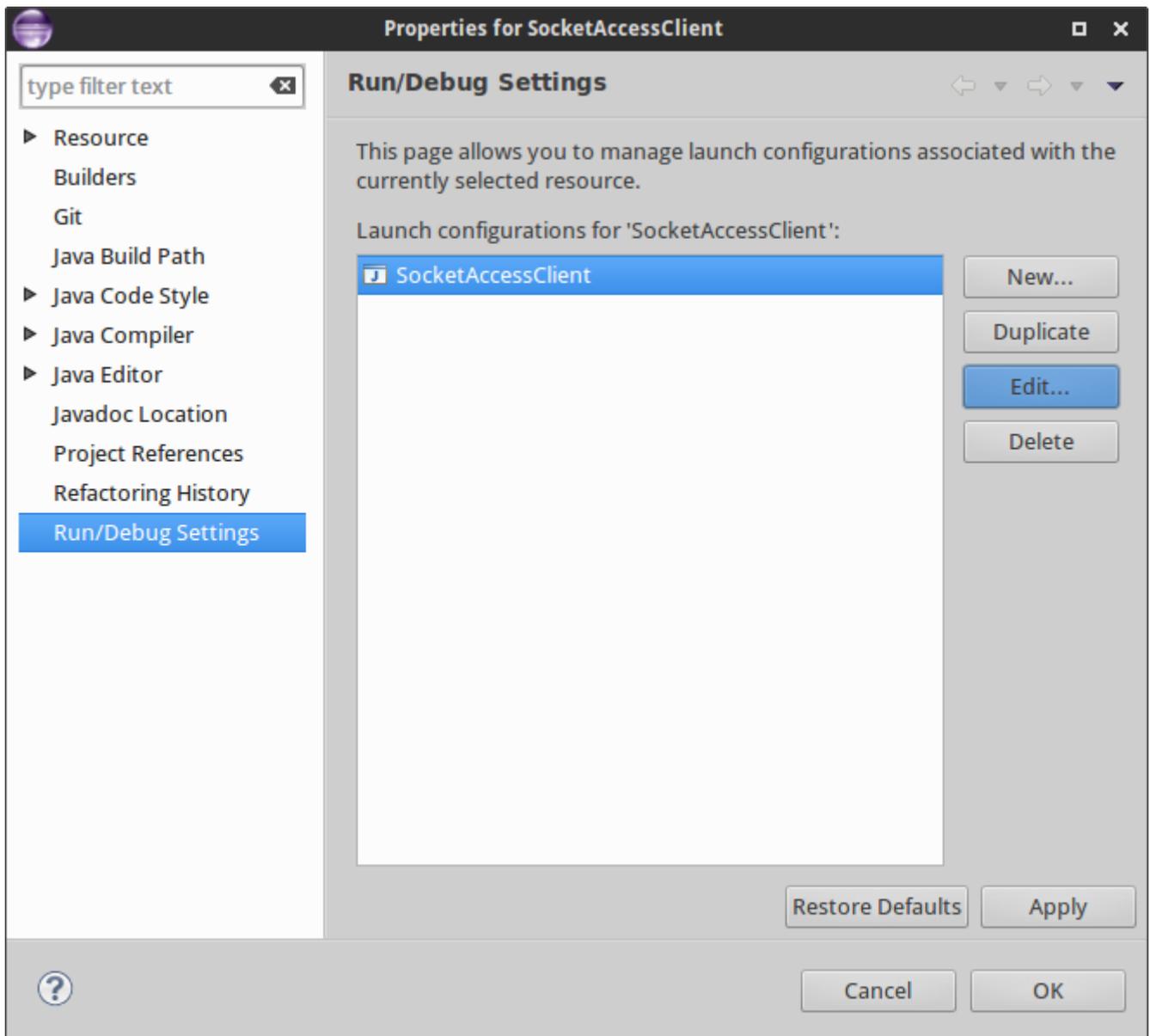
```
java -jar client.jar npe_ip_address port
```

Where:

- npe_ip_address is address of Your npe unit
- port is port on which our SocketAccess channel in iMod is listening. Use the same port value as in configuration of SocketAccess channel.

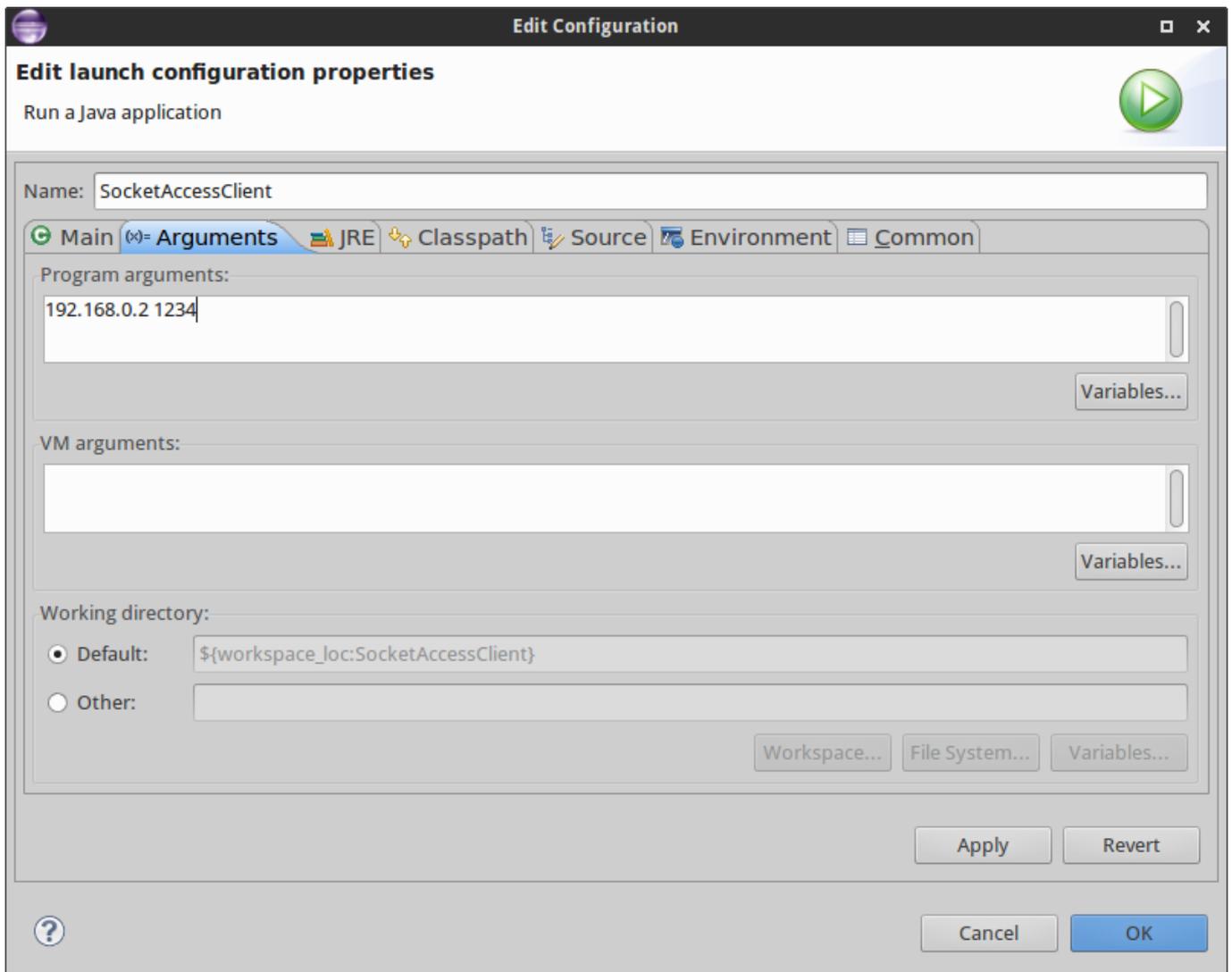
When running client in eclipse ip address and port values will be taken from default values from code. To use another values change defaults in code or set arguments in eclipse:

- Go to project's properties.
- Select *Run/Debug Settings*
- Select our client project and click *Edit...*



If there are no positions in this window, close it and run project and then return to this window.

- In *Arguments* tab type custom arguments in *Program arguments* section (ip_address port)



Using client



This implementation of server can serve only one client at the same time. Other clients will behave as they would be connected but server wouldn't serve them (there won't be any response from server to that clients).

After the client ran simple type commands:

- `read:property_number` to read value of property i.e. `read:100`
- `write:property_number:value` to write new value to specified property i.e. `write:100:1`
- `exit` - to close connection and exit client. After `exit` another client can connect to `SocketAccessServer`

Any others commands will be refused by server and resend to client with appropriate comment.



If client doesn't connect or other error appears, program will throw exception and display it in console.



If client was closed in other way than *exit* command, sometimes server won't serve other client. In this case, new client won't receive any messages. To fix it restart iMod.

[Appendix](#)

[iMod SDK Libraries](#)

[iMod SDK Examples](#)