TECHBASE
AUTOMATYKA ELEKTRONIKA INFORMATYKA PRZEMYSŁOWA

Seria urządzeń

iMod

Opis SDK aplikacji iMod

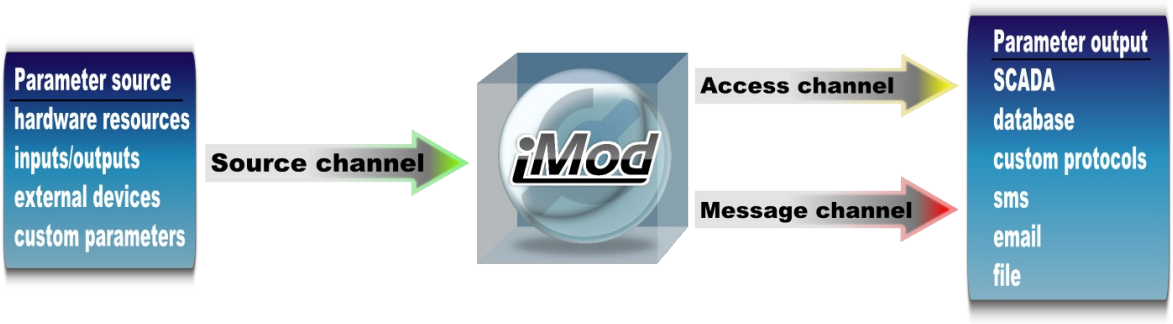
Spis treści

Rozdział 1 Ogólne informacje

Dokument powstał w celu opisania sposobu wykorzystania dostępnego SDK aplikacji iMod.

Aplikacja iMod wykorzystuje trzy główne typy kanałów:

- kanał źródłowy – typu Source-Channel
- kanał dostępu – typu Access-Channel
- kanał informacyjny – typu Messege-Channel



Niezależnie od typu – implementacja kanału wymaga stworzenia trzech głównych klas.

1. Activator
2. ChannelFactorySrv
3. ChannelImpl

Nazwa klas tworzona jest w konwencji [nazwa_kanału][typ_kanału][nazwa_klasy] gdzie:

- nazwa_kanału – dowolny ciąg znaków
- typ_kanału – pierwsza litera typu kanału (A – Access ,S – source ,M – messege)
- nazwa_klasy – nazwa implementowanej klasy (Activator, FactorySrv, ChannelImpl)

Klasa Activator

Klasa Activator implementowanego kanału musi dziedziczyć po klasie CommonActivator.

Metody wymagające implementacji:

TemplateMActivator()

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

start()

Metoda odpowiedzialna jest za stworzenie i uruchomienie instancji ChannelFactorySrv.

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

Klasa ChannelFactorySrv

Klasa FactorySrv niezależnie od kanału musi implementować interfejs ISChannelFactory. Metody wymagające implementacji

IChannel newInstance(IChannelProxy iModEngine, boolean scanmode)

Metoda odpowiedzialna jest za stworzenie nowej instancji implementowanego kanału i dodanie jej do listy dostępnych kanałów.

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

close()

Metoda odpowiedzialna jest za zamknięcie wszystkich utworzonych instancji implementowanego kanału.

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

Klasa ChannelImpl

Klasa ChannelImpl niezależnie od kanału musi implementować interfejs IChannel oraz dziedziczyć po jednej z klas w zależności od typu kanału:

- CommonMChannel
- CommonAChannel
- CommonSChannel

Metody, które należy zaimplementować:

TemplateMChannelImpl

Metoda odpowiedzialna jest za uruchomienie kanału.

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

getParameter

Metoda odpowiedzialna jest za obsługę zapytania typu getParameter, które wykorzystywane jest przez aplikację iMod do odczytania danego parametru z kanału.

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

List<Error> getStatus()

Metoda zwraca do aplikacji iMod informację o statusie kanału.

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

void setConfig(IProperties arg0)

Metoda umożliwia ustawienie dodatkowych parametrów konfiguracyjnych kanału

```
public void setConfig(IProperties p) {

	}
```

Status setParameter(String id, Object val)

Metoda wykorzystywana jest przez aplikację iMod do wymuszenia zmiany wartości parametru w danym kanale.

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

refreshParameters()

Medoda wykorzsytywana do odświeżenia wszystkich dostępnych parametrów danego kanału.

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

configure()

Metoda wykorzystywana jest do początkowej konfiguracji kanału – przykładowo do uporządkowania parametrów w celu poprawy wydajności.

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

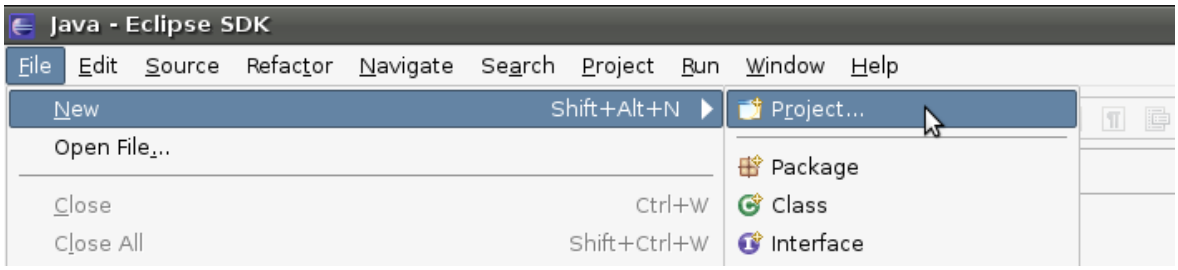
Rozdział 2 Przygotowanie środowiska Eclipse

Rozdział drugi zawiera opis procesu przygotowania środowiska w celu wykorzystania iMod SDK w środowisku Eclipse IDE.

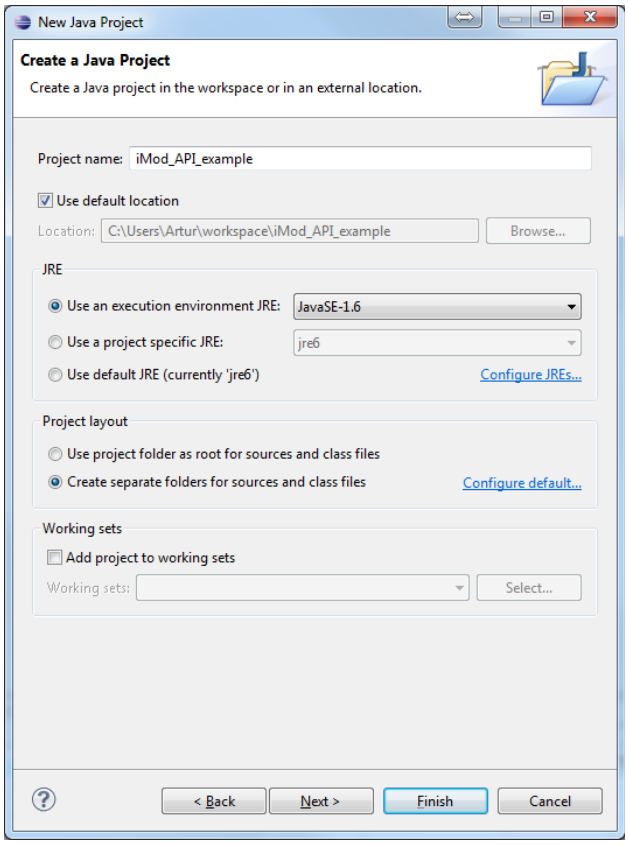
W tym rozdziale wykorzystujemy wersję programu - `Eclipse Classic`.
Wszystkie opisane poniżej pliki wraz z przykładowym *workspace* i *projektem* dostępne są w archiwum na serwerze FTP: *ftp.a2s.pl*

2.1. Stworzenie projektu

Proces tworzenia rozpocznij od uruchomienia programu Eclipse i podania ścieżki do *Workspace,* gdzie będą przechowywane utworzone projekty. Na początku stwórz projekt.



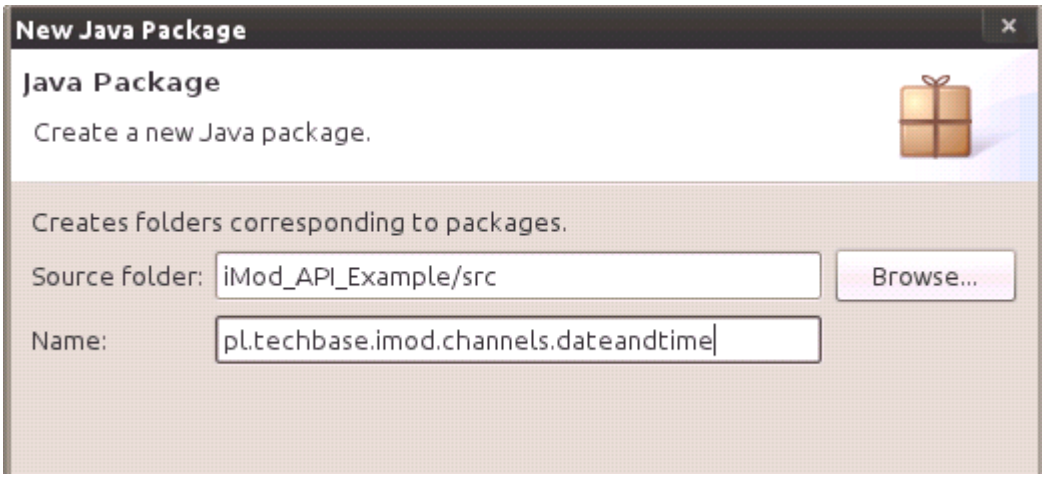
W kolejnym oknie wybierz typ projektu: *Java Project* i kliknij *Next.*
Następnie zostaniemy poproszony o podanie nazwy projektu – wpisz *iMod_API_example* W polu *Directory* znajduje się ścieżka do katalogu z Twoim projektem. Potwierdzając kliknij *Finish*.



2.2. Dodanie JAVA PACKAGE

1. Dodaj *package* zgodnego z wymaganiami SDK (File → new → package):

```
pl.techbase.imod.channels.nazwa
```



2. Następnie w głównym katalogu projektu utwórz podkatalog *lib,* do którego skopiuj plik *imodlib.jar*

Plik znajduje się w podkatalogu /jar/*protocols* katalogu instalacyjnego aplikacji iMod.

3. Kliknij na dodany plik *imodlib.jar* prawym przyciskiem myszy i wybierz (Build Path → Add to build path)

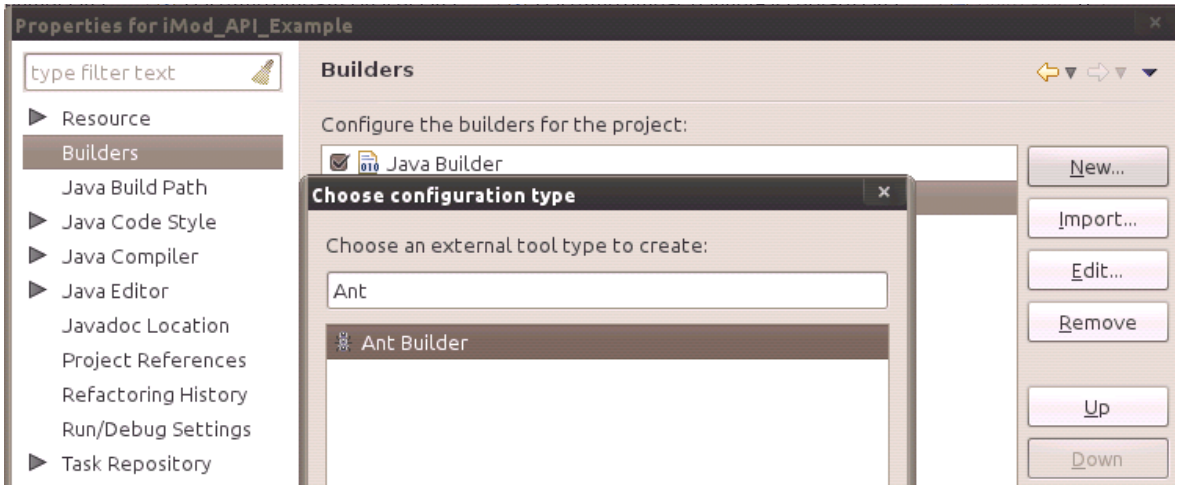
4. Po utworzeniu package oraz dodaniu pliku SDK do katalogu głównego projektu dodaj również plik *build.xml*, który będzie odpowiedzialny za kompilację oraz tworzenie archiwum *jar*.

Zawartość pliku *build.xml*, który zostanie wykorzystany przy tworzeniu kanału DateAndTime*:*

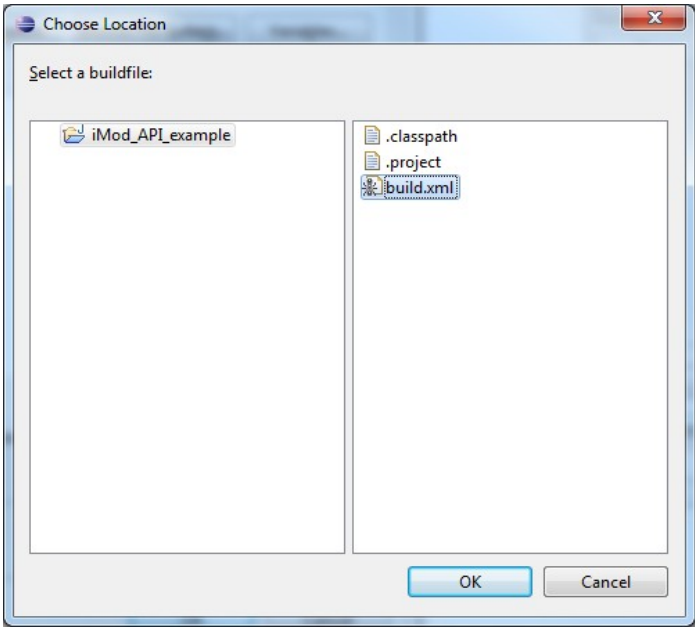
```
<?xml version="1.0" encoding="iso-8859-2"?>
<project name="Example date plugin for iMod - create package" basedir="."
default="make_date_jar">

<property name="techbase.name" value="Customer" />
<property name="build.dir" location="builder/build/" />
<property name="class.dir" location="bin/pl" />
<property name="main.class"
value="pl.techbase.imod.channels.dateandtime.DateAndTimeSActivator" /> <!--
put here your class name -->

	<target name="clean">
		<delete dir="${build.dir}"/>
	</target>

<!-- ////////////////////////////prepare////////////////////////////////////
-->
	<target name="prepare" depends="clean">
		<mkdir dir="${build.dir}" />
	</target>

<!-- ////////////////////////////make_jar/////////////////////////////  -->
	<target name="make_date_jar" depends="prepare">
		<jar destfile="${build.dir}/date.jar">
		<fileset dir="bin">
			<include name="pl/**/*.class"/>
		</fileset>
		<manifest>
			<attribute name="Built-By" value="${techbase.name}"/>
			<attribute name="Main-Class" value="${main.class}"/>
			<attribute name="Class-Path" value="imodlib.jar" />
		</manifest>
		</jar>

	</target>

</project>
```

Ważne aby parametr <*property name*="*main.class*" *value*="" /> wskazywał pełną ścieżkę do klasy implementującej kanał, np. *DateAndTimeSActivator*

Po utworzeniu pliku w ustawieniach projektu dodaj nowe narzędzie budowania poprzez wybranie *Builders>New> Ant Builder* i wskazanie pliku *build.xml*.



Po poprawnym utworzeniu narzędzia budującego możesz przejść do najważniejszego etapu, którym będzie implementacja przykładowego kanału.



Rozdział 3 Implementacja przykładowego kanału typu Source-Channel

Rozdział trzeci zawiera opis przykładowej implementacji kanałów komunikacyjnych dedykowanych do SDK aplikacji iMod. W celu przedstawienia struktury kanału typu Source-Channel stworzysz kanał umożliwiający odczytanie oraz modyfikację daty na urządzeniu iMod.

Podczas budowania swojego kanału należy wykorzystać biblioteki z urządzenia iMod. Przydatne są takie biblioteki jak:

- javolution-5.5.1.jar (miejsce instalacji pakietu JAVA)
- commons-logging-1.1.1.jar (.../iMod/jar/libs)
- log4j-1.2.16.jar (.../iMod/jar/libs)

Do poniższego przykładu, wystarczy zaimportować dodatkowo commons-logging oraz log4j.

Dla potrzeb tego dokumentu zostanie on nazwany kanałem: *DateAndTime*

3.1. Zdefiniowanie struktury pliku MainConfig.xml

Proces implementacji rozpocznij od zdefiniowania struktury XML, która będzie opisywała tworzony kanał. Dla potrzeb tego dokumentu przyjmujemy poniższe struktury XML.

Definicja kanału w pliku XML

```
<source-channel name="NpeDate">
       <protocol name="DateAndTime"/>
       <port>"DayAndTimePort"</port>
       <property name="DateAndTimeProperty1" value="true"/>
       <property name="DateAndTimeProperty2" value="false"/>
       <cycle>60s</cycle>
 </source-channel>
```

Definicja parametrów w pliku XML

```
<parameter type="int16">
       <id>"Hour"</id>
       <description>"return NPE hour"</description>
       <source-channel channel-name="NpeDate" parameter-id="hour"/>
       <access-channel channel-name="Modbus_S1" parameter-id="1"/>
</parameter>

<parameter type="int16">
       <id>"Minute"</id>
       <description>"return NPE minute"</description>
       <source-channel channel-name="NpeDate" parameter-id="minute"/>
       <access-channel channel-name="Modbus_S1" parameter-id="2"/>
</parameter>

<parameter type="int16">
       <id>"seconds"</id>
       <description>"return NPE seconds"</description>
       <source-channel channel-name="NpeDate" parameter-id="second"/>
       <access-channel channel-name="Modbus_S1" parameter-id="3"/>
```

```
</parameter>

<parameter type="int16">
       <id>"Year"</id>
       <description>"return NPE year"</description>
       <source-channel channel-name="NpeDate" parameter-id="year"/>
       <access-channel channel-name="Modbus_S1" parameter-id="3"/>
</parameter>



<parameter type="int16">
       <id>"Month"</id>
       <description>"return NPE Month"</description>
       <source-channel channel-name="NpeDate" parameter-id="month"/>
       <access-channel channel-name="Modbus_S1" parameter-id="3"/>
</parameter>

<parameter type="int16">
       <id>"Day"</id>
       <description>"return num. of day in the month"</description>
       <source-channel channel-name="NpeDate" parameter-id="day"/>
       <access-channel channel-name="Modbus_S1" parameter-id="3"/>
</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>
```

3.2. Klasa Activator

Implementacja prostego kanału nie wymaga dodatkowych modyfikacji klasy *TemplateSActivator* – jedynie należy zmienić nazwę klasy na *DateAndTimeSActivator*.

Stwórz klasę *DateAndTimeSActivator* o zawartości:

```
package pl.techbase.imod.channels.dateandtime;
import pl.techbase.imod.imodlib.common.CommonActivator;

public class DateAndTimeSActivator extends CommonActivator {

	public DateAndTimeSActivator()
	{
		super();
	}

	@Override
	public void start() {
		this.serviceFactory = new DateAndTimeSChannelFactorySrv(); // you
will create this class in next step
	}
}
```

3.3. Klasa ChannelFactorySrv

Również klasa *DateAndTimeSChannelFactorySrv* nie wymaga dodatkowych modyfikacji względem klasy *TemplateSChannelFactorySrv*– z wyjątkiem zmian nazw klas.

Stwórz klasę *DateAndTimeSChannelFactorySrv* o zawartości:

```
package pl.techbase.imod.channels.dateandtime;
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 DateAndTimeSChannelFactorySrv implements ISChannelFactory {

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

	@Override
	public IChannel newInstance(IChannelProxy iModEngine, boolean scanmode)
{
		IChannel channel = new
DateAndTimeSChannelImpl(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();
		}
	}
}
```

3.4. Klasa ChannelImpl

Docelowa implementacja kanału realizowana jest w klasie *DateAndTimeSChannelImpl*, która udostępnia zasoby danego źródła do iModEngine.
Stwórz klasę `DateAndTimeSChannelImpl` o poniższej treści:

```
public class DateAndTimeSChannelImpl 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(DateAndTimeSChannelImpl.class);

}
```

Klasa LOG nie będzie widoczna do momentu aż zaimportujesz plik log4j.

DateAndTimeSChannelImpl(IChannelProxy iModEngine, boolean scanmode)

Metoda *DateAndTimeSChannelImpl* odpowiedzialna jest za uruchomienie kanału, dodatkowo zawiera informację wpisywaną do logu aplikacji w celu upewnienia się czy klasa została uruchomiona poprawnie.

Dodani poniższą metodę:

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

getParameter(String id)

Metoda *getParameter* odpowiedzialna jest za obsługę zapytania typu getParameter, które wykorzystywane jest przez aplikację iMod do odczytania danego parametru z kanału. Ważne aby metoda zwracała odczytaną wartość poprzez *return*.

Dodatkowo należy zaktualizować wewnętrzną bazę iMod'a o odczytaną wartość poprzez metodę *this.m_channel_proxy.getParameter(id).setValue(new_value)* a następnie zwrócić informację o statusie parametru do iModEngine poprzez metodę *setStatus*

W tworzonym przykładzie metoda powinna zawierać mechanizm rozróżnienia rodzaju atrybutu i zwróci odpowiednią składową daty w urządzeniu.

Dodaj do klasy poniższy obiekt:

```
public Object getParameter(String id) {
	int new_value = 0;
	log.info("Try to getParameter: "+id);
	cal = Calendar.getInstance();
if(this.m_channel_proxy.getParamSpecs().get(id).keySet().contains("source-
channel.parameter-id"))
{String value = this.m_channel_proxy.getParamSpecs().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(Status.ACTIVE.toString());
			this.m_channel_proxy.getParameter(id).setValue(new_value);
			return Integer.valueOf(new_value);
		}
```

getStatus()

Metoda *getStatus* zwraca do aplikacji iMod informację o statusie kanału. Dodaj ją do klasy:

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

setConfig(IProperties arg0)

Metoda umożliwiająca ustawienie dodatkowych parametrów konfiguracyjnych kanału to *setConfig*:

```
public void setConfig(IProperties p) {

	}
```

setParameter(String id, Object val)

Metoda *setParamter* wykorzystywana jest przez aplikację iMod do wymuszenia zmiany wartości parametru w danym kanale.

Dodaj poniższą treść do klasy:

```
public Status setParameter(String id, Object val) {
if (this.m_channel_proxy.getParamSpecs().get(id).keySet().contains("source-
channel.parameter-id")) {

String value = this.m_channel_proxy.getParamSpecs().get(id).get("source-
channel.parameter-id").get(0);

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.ACTIVE;
}
```

refreshParameters()

Metoda wykorzystywana jest do odświeżenia wszystkich zdefiniowanych parametrów przypisanych do danego kanału.

W przypadku prostego kanału metoda refreshParameters będzie zawierać pętlę odczytującą poszczególne parametry za pomocą metody *getParameter(id)*

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

configure()

Metoda *configure* wykorzystywana jest do początkowej konfiguracji kanału. W przypadku kanału DateAndTime metoda odpowiedzialna będzie za wyłuskanie odpowiednich parametrów początkowych z pliku XML oraz do uporządkowania parametrów w celu poprawy wydajności.

Metoda *configure* wykonywana jest jednorazowo podczas inicjalizacji kanału - tutaj należałoby zawrzeć wszelkie skalowania i inne dopasowania parametrów - w tym dodanie parametrów do listy tak aby przy uruchomieniu metody np *refreshParameters()* od razu odczytywać parametry - poprawi to wydajność - szybkość odczytu.

Przykładowa definicja kanału w pliku *MainConfig*.xml:

```
<source-channel name="NpeDate">
      <protocol name="DateAndTime"/>
      <port>"DayAndTimePort"</port>
      <property name="DateAndTimeProperty1" value="true"/>
      <property name="DateAndTimeProperty2" value="false"/>
      <cycle>60s</cycle>
</source-channel>
```

W celu pokazania sposobu wyłuskiwania parametrów typu *property* z definicji kanału, dodane zostały *DateAndTimeProperty1* i *DateAndTimeProperty2*, których wartości zostaną dodane do logu aplikacji iMod.

Dodaj przykładową implementacę metody *configure()*:

```
protected boolean configure() throws IModException {
      boolean ret;
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.DateAndTimeProperty1")){
            log.info("DateAndTimeProperty1 has value: "+ nextValue);
      }else if(nextKey.equals("property.DateAndTimeProperty2"))
      {
            log.info("DateAndTimeProperty2 has value: "+ nextValue);
      }
}}}
return super.configure();
}
}
```

W metodzie *configure()* pobierasz i przeglądasz sparsowaną zawartość pliku MainConfig.xml.

Sprawdzanie nazwy znacznika odbywa się poprzez metodę *nextKey.equals("nazwa"):*

```
String nextKey = (String)iterKey.next();
String nextValue = (String) prop.get(nextKey);
       if (nextKey.equals("port")) {
(..) /* Obsługa danego atrybutu */
       }
```

Jeżeli napotkamy przykładowo *<port></port>* to za pomocą metody *prop.get(nextKey)* możemy pobrać ciąg znaków, wpisany pomiędzy znacznikami port.

Następnie w zależności od wymagań kanału – należy odpowiednio zinterpretować wpisany ciąg. Przykładowo za pomocą *Pattern i Matcher* można sprawdzić i wyłuskać interesujące informacje, np. numer portu TCP lub portu szeregowego oraz inne informacje takie jak Baudrate itp.

Ciągi znaków wpisane w znacznikach definiujących kanał muszą być odpowiednio definiowane i interpretowane w odniesieniu do danego kanału.

3.5. Modyfikacja manifestu imodEngine

Po zakończeniu właściwej implementacji kanału DateAndTime i utworzeniu archiwum JAR zmodyfikuj manifest iModEngine aby jądro aplikacji widziało klasy nowego kanału.

Pierwszym krokiem jest pobranie z urządzenia pliki imodTiger.jar. Najwygodniej dodać plik do głównego katalogu projektu w Eclipse.

Plik imodTiger.jar znajduje się w głównym katalogu instalacyjnym aplikacji iMod.

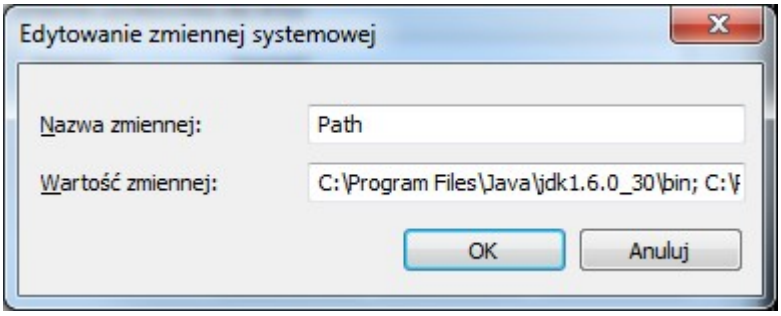
Docelowe miejsce można sprawdzić za pomocą polecenia:

getenv IMOD_VERSION

Po pobraniu pliku wypakuj plik manifestu za pomocą polecenia:

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

W systemie windows7, jeśli polecenie JAR jest nie widoczne, należy dodać na początku zmiennej systemowej odnośnik do katalogu /bin



Następnie dodaj do wiersza Class-Path wpis z miejscem docelowym pliku date.jar

```
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 ../j ar/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/post gresql-8.4-703.jdbc4.jar
../jar/protocols/date.jar
```

Następnie za pomocą plecenia aktualizujemy manifest w pliku imodTiger:

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

Po zakończeniu aktualizacji należy wgrać nowy plik imodTiger.jar do głównego katalogu aplikacji iMod na urządzeniu NPE.

Nowo utworzony plik date.jar implementujący kanał DateAndTime należy wgrać do katalogu wskazanego w Manifeście (jar/protocols).

Po wgraniu plików można do pliku MainConfig.xml dodać parametry zdefiniowane dla nowego kanału w celu sprawdzenia poprawności działania.