TECHBASE
WE MAKE CONNECTIONS. SIMPLE

Serie de dispositivos

iMod

Descripción del SDK de la aplicación iMod

Índice

Capítulo 1 Información general

El documento ha sido creado para describir el modo de uso del SDK disponible para la aplicación iMod. La aplicación iMod utiliza los tres principales tipos de canales:

- canal de fuente– tipo source-channel
- canal de acceso– tipo Access-Channel
- canal de información – tipo Messege-Channel

Independiente del tipo - para implementar un canal hay que crear tres clases principales

1. Activator
2. ChannelFactorySrv
3. ChannelImpl

El nombre de la clase se compone de los siguientes elementos:
[nombre_del_canal][tipo_del_canal][nombre_de_la_clase] donde:

- el nombre_del_canal – es cualquier serie de caracteres
- tipo_del_canal – primera letra del tipo de canal (A – Access ,S – source, M – message)
- nombre_de_la_clase – nombre de la clase que se está implementando (Activator, FactorySrv, ChannelImpl)

Clase Activator

La clase Activator del canal que se está implementando debe heredar de la clase CommonActivator. Los métodos que requieren implementación:

TemplateMActivator()

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

start()

Es un método responsable de la creación y ejecución de la instancia ChannelFactorySrv.

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

Clase ChannelFactorySrv

Independiente del canal, la clase FactorySrv debe implementar una interfaz ISChannelFactory. Los métodos que requieren implementación

IChannel newInstance(IChannelProxy iModEngine, boolean scanmode)

Es un método que sirve para crear una nueva instancia en el canal que se está implementando y añadirla en la lista de los canales disponibles.

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

close()

Es un método responsable de cerrar todas las instancias creadas en el canal que se está implementando.

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

Clase ChannelImpl

Independiente del canal, la clase ChannelImpl debe implementar la interfaz IChannel y heredar de una de las clases, en función del tipo del canal:

- CommonMChannel
- CommonAChannel
- CommonSChannel

Los métodos que deben implementarse son:

TemplateMChannelImpl

Es un método responsable de la activación del canal.

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

getParameter

Es un método responsable de la gestión de la consulta tipo getParameter que es usada por la aplicación iMod para leer un determinado parámetro del canal.

```
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()

Este método devuelve a la aplicación iMod la información sobre el estado del canal.

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

void setConfig(IProperties arg0)

Este método permite determinar los parámetros de configuración del canal.

```
public void setConfig(IProperties p) {

	}
```

Status setParameter(String id, Object val)

Este método es usado por la aplicación iMod para forzar un cambio de valor del parámetro en un canal en concreto.

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

refreshParameters()

Este método es usado para actualizar todos los parámetros disponibles en un canal en concreto.

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

configure()

Este método es usado para la configuración inicial del canal - por ejemplo para ordenar los parámetros con el fin de mejorar la eficiencia.

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

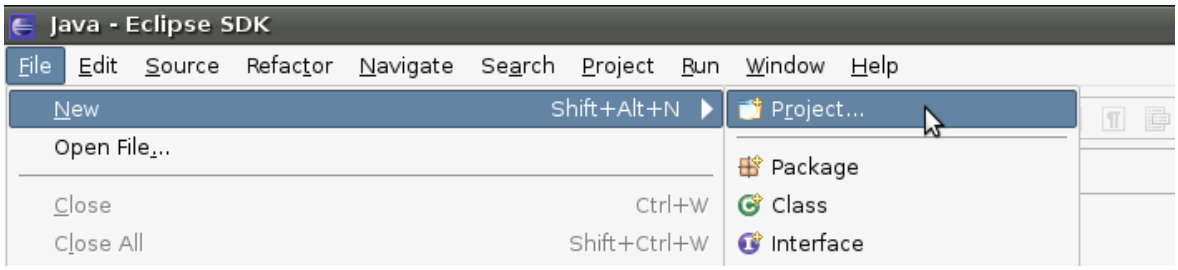
Capítulo 2 Preparación del entorno Eclipse

El segundo capítulo contiene la descripción del proceso de preparación del entorno para usar el iMod SDK en el entorno Eclipse IDE.

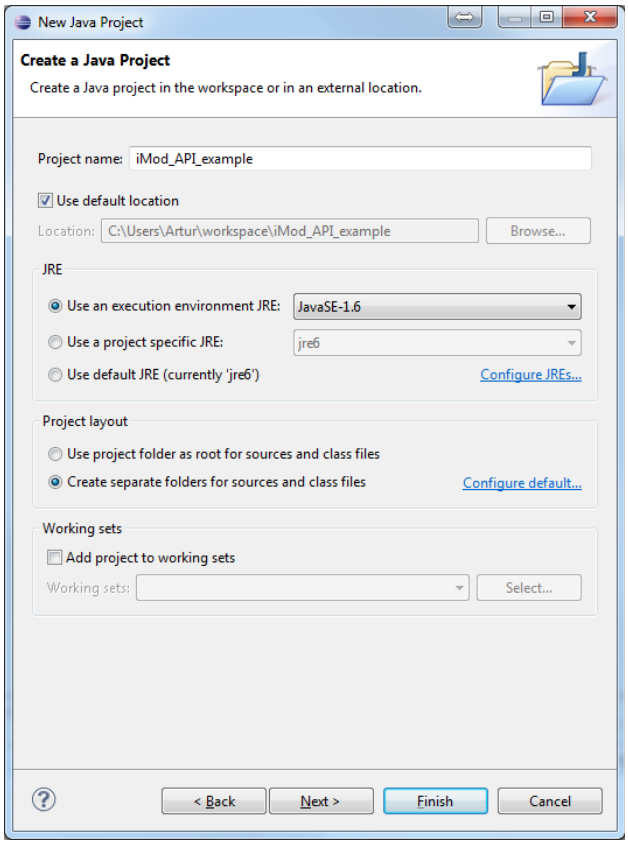
La versión del programa que usamos en este capítulo es la `Eclipse Classic`.
Todos los archivos descritos a continuación junto con el ejemplo de workspace y proyecto están disponibles en un archivo en el servidor FTP: *ftp.a2s.pl*

2.1. Creación de un proyecto

Empieza el proceso de creación del proyecto por iniciar el programa Eclipse e indicar la ruta al *Workspace*, donde se guardarán los proyectos que se van a crear. Al principio crea un proyecto.



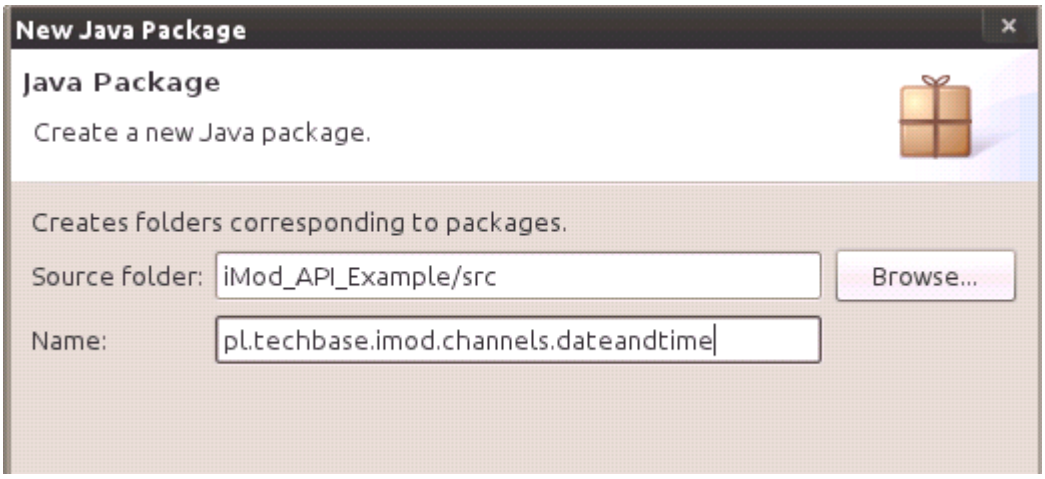
En la siguiente ventana selecciona el tipo de proyecto: *Java Project* y haz clic en *Next*. A continuación el programa te solicitará que introduzcas el nombre del proyecto – escribe *iMod_API_example*. En el campo *Directory* está la ruta de acceso al catálogo con tu proyecto. Para confirmar haz clic en *Finish*.



2.2. Adición de JAVA PACKAGE

1. Añade un *package* compatible con los requisitos del SDK (File → new → package):

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



2. A continuación en el catálogo principal del proyecto crea un subcatálogo *lib*. Copia el archivo *imodlib.jar* y pégalo en este subcatálogo.

El archivo está en el subcatálogo */jar/protocols* del catálogo de instalación de la aplicación iMod.

3. Haz clic en el archivo *imodlib.jar* que acabas de añadir con el botón derecho del ratón y selecciona (Build Path → Add to build path)

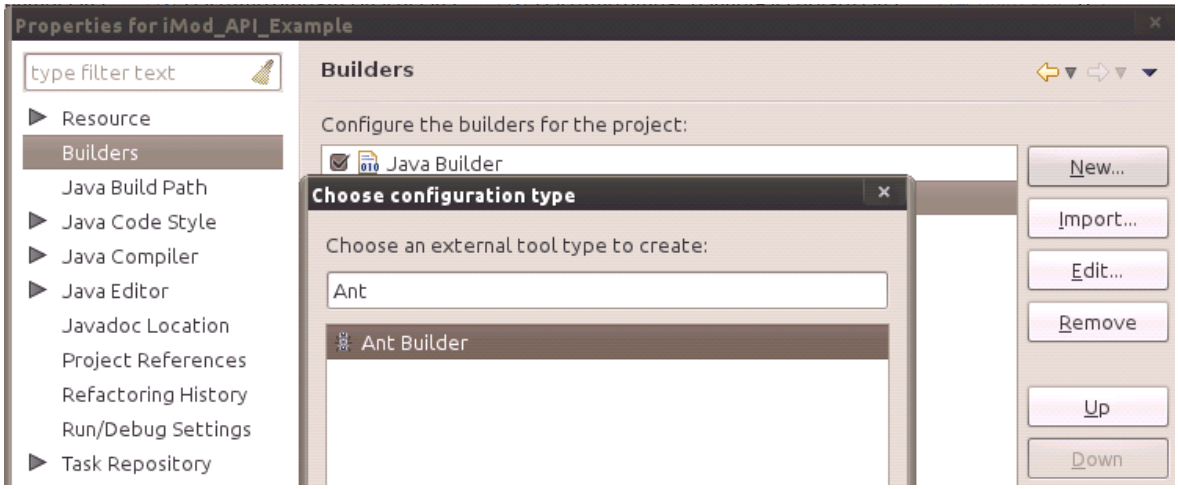
4. Una vez creado el package y añadido el archivo SDK al catálogo principal del proyecto, añade también el archivo *build.xml*, que será responsable de la compilación y creación del archivo *jar*.

Este es el contenido del archivo *build.xml*, que se usará a la hora de crear el canal DateAndTime:

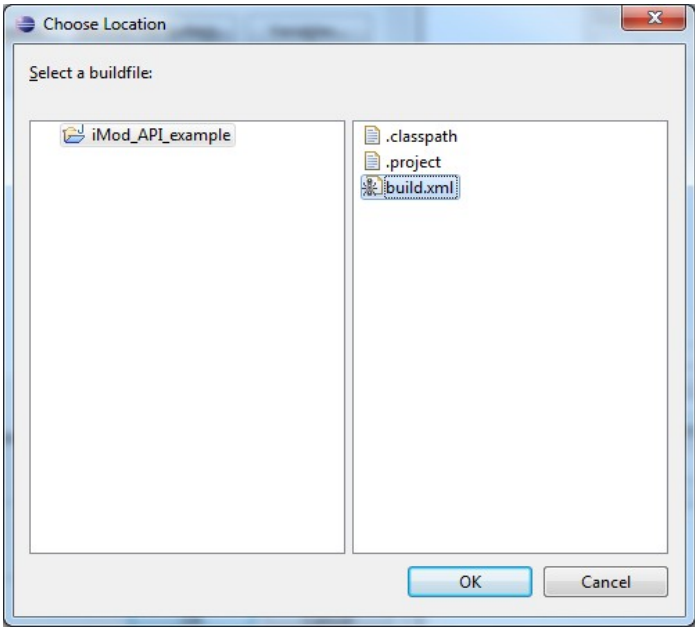
```
<?xml version="1.0" encoding="iso-8859-2"?>
<project name="Example date plugin for iMod - create package" basedir="."
default="make_date_jar">

<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>
```

Es importante que el parámetro *<property name="main.class" value="" />* indique la ruta completa a la clase que implemente el canal, por ej. *DateAndTimeSActivator*

Una vez creado el archivo, añade en la configuración una nueva herramienta de construcción (builder) seleccionando *Builders>New> Ant Builder* e indicando el archivo *build.xml*.



Después de crear correctamente la herramienta de construcción puedes pasar a la etapa más importante, es decir a la implementación de un canal ejemplar.



Capítulo 3 Implementación de un ejemplo de canal tipo Source-Channel

En el tercer capítulo presentamos un ejemplo de implementación de canales de comunicación dedicados para el SDK de la aplicación iMod. Para presentar la estructura del canal tipo Source-Channel vas a crear un canal que permite la lectura y modificación de la fecha en el dispositivo iMod.

Durante la construcción de un nuevo canal hay que usar las bibliotecas del dispositivo iMod. Se pueden aprovechar tales bibliotecas como:

- javolution-5.5.1.jar (lugar de instalación del paquete JAVA)
- commons-logging-1.1.1.jar (.../iMod/jar/libs)
- log4j-1.2.16.jar (.../iMod/jar/libs)

En el caso del ejemplo presentado a continuación, basta con importar commons-logging y log4j.

Para las necesidades del presente documento lo llamaremos el canal: *DateAndTime*

3.1. Definición de la estructura del archivo MainConfig.xml

Empieza el proceso de implementación por definir la estructura del XML, que describirá el canal que estás creando. Para las necesidades del presente documento vamos a suponer las siguientes estructuras del XML.

Definición del canal en el archivo 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>
```

Definición de los parámetros en el archivo 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. Clase Activator

Para implementar un canal simple no es necesario hacer modificaciones adicionales de la clase *TemplateSActivator* – sólo hay que modificar el nombre de la clase a *DateAndTimeSActivator*.

Crea la clase *DateAndTimeSActivator* con este contenido:

```
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. Clase ChannelFactorySrv

Tampoco la clase *DateAndTimeSChannelFactorySrv* requiere modificaciones adicionales con respecto a la clase *TemplateSChannelFactorySrv* – excepto los cambios de los nombres de las clases.

Crea la clase *DateAndTimeSChannelFactorySrv* con el siguiente contenido:

```
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. Clase ChannelImpl

La prevista implementación del canal se realiza en la clase *DateAndTimeSChannelImpl* que facilita el acceso a los recursos de una determinada fuente para iModEngine.
Crea la clase `DateAndTimeSChannelImpl` con el siguiente contenido:

```
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);

}
```

La clase LOG no será visible hasta que importes el archivo log4j.

DateAndTimeSChannelImpl(IChannelProxy iModEngine, boolean scanmode)

El método *DateAndTimeSChannelImpl* es responsable de la activación del canal, además contiene la información que se incluirá en el log de la aplicación para asegurarse de que la clase haya sido activada correctamente.

Añade el siguiente método:

```
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)

El método *getParameter* es responsable de la gestión de la consulta tipo getParameter, que es usada por la aplicación iMod para leer un determinado parámetro de un canal. Es importante que el método devuelva el valor leído con el uso de *return*.

Además, hay que actualizar la base interna del iMod con el valor leído con el uso del método *this.m_channel_proxy.getParameter(id).setValue(new_value)* y a continuación devolver la información sobre el estado del parámetro a iModEngine con el uso del método *setStatus*.

En este ejemplo el método debe incluir el mecanismo de diferenciación de tipo del atributo y devolverá el respectivo componente de la fecha en el dispositivo.

Añade a la clase el siguiente objeto:

```
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()

El método *getStatus* devuelve a la aplicación iMod la información sobre el estado del canal. Añádelo a la clase:

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

setConfig(IProperties arg0)

Un método que permite el ajuste de los parámetros de configuración adicionales del canal es *setConfig*:

```
public void setConfig(IProperties p) {

	}
```

setParameter(String id, Object val)

El método *setParamter* es usado por la aplicación iMod para forzar el cambio del valor del parámetro en un determinado canal.

Añade el siguiente contenido a la clase:

```
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()

Este método es usado para actualizar todos los parámetros definidos que han sido asignados a un determinado canal.

En caso de un canal simple, el método refreshParameters contendrá un bucle que leerá los particulares parámetros con el uso del método *getParameter(id)*

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

configure()

El método *configure* es usado para la configuración inicial del canal. En el caso del canal DateAndTime, el método será responsable de extraer los respectivos parámetros iniciales del archivo XML y ordenar los parámetros con el fin de mejorar la eficiencia.

El método *configure* se ejecuta una vez al iniciar el canal – aquí es donde deberían incluirse todos los ajustes de escala y otras adaptaciones de los parámetros - incluida la adición de los parámetros a la lista para que al mismo tiempo, cuando se inicie el método, por ej. *refreshParameters()* también se lean los parámetros - esto mejoraría la eficiencia - la rapidez de lectura.

Ejemplo de la definición del canal en el archivo *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>
```

Para presentar el modo de extracción de los parámetros tipo property de la definición del canal, han sido añadidos *DateAndTimeProperty1* y *DateAndTimeProperty2*, cuyos valores serán incluidos en el log de la aplicación iMod.

Añade un ejemplo de implementación del método *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();
}
}
```

En el método *configure()* obtienes y consultas el contenido parseado del archivo MainConfig.xml.

La verificación del nomrbe de la etiqueta se realiza con el uso del método *nextKey.equals("nombre")*:

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

Si encontremos por ejemplo *<port></port>*, entonces con el uso del método *prop.get(nextKey)* podemos obtener una serie de caracteres, inscrita entre las etiquetas port.

A continuación, en función de los requisitos del canal, hay que interpretar adecuadamente esta serie. Por ejemplo con el uso de *Pattern* y *Matcher* se puede verificar y extraer la información que nos interesa, por ej. el número del puerto TCP o del puerto serie u otra información como Baudrate etc.

Las series de caracteres inscritas entre las etiquetas que definen el canal deben ser adecuadamente definidas e interpretadas con respecto a un determinado canal.

3.5. Modificación del manifiesto imodEngine

Una vez terminada la implementación adecuada del canal DateAndTime y una vez creado el archivo JAR, modifica el manifiesto iModEngine para que el núcleo de la aplicación pueda ver las clases del nuevo canal.

El primer paso consiste en obtener los archivos *imodTiger.jar* desde el dispositivo. Lo más fácil es añadir el archivo en el catálogo principal del proyecto en Eclipse.

El archivo *imodTiger.jar* se encuentra en el catálogo principal de instalación de la aplicación iMod.

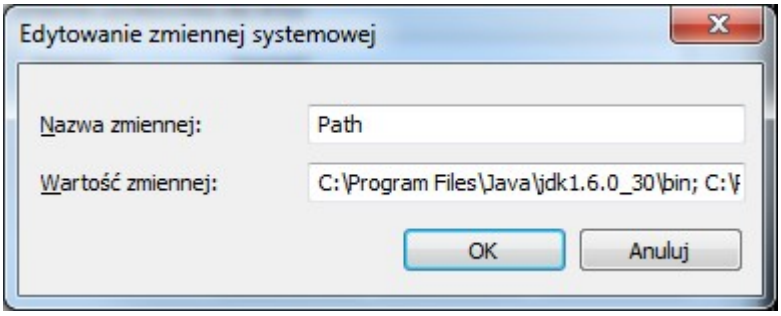
El lugar de destino puede ser comprobado con el uso del comando:

getenv IMOD_VERSION

Una vez obtenido el archivo, desempaqueta el archivo de manifiesto con el uso del comando:

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

En el sistema Windows7, si el comando JAR no es visible, hay que añadir un enlace al catálogo */bin* al principio de la variable de entorno.



Después, añade en la línea Class-Path la información sobre el lugar de destino del archivo *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
```

Después, con el uso del comando actualizamos el manifiesto en el archivo imodTiger:

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

Una vez terminada la actualización, hay que cargar el nuevo archivo *imodTiger.jar* al catálogo principal de la aplicación iMod en el dispositivo NPE.

Hay que cargar el nuevo archivo *date.jar* que implementa el canal DateAndTime al catálogo indicado en el Manifiesto (jar/protocols).

Una vez cargados los archivos, es posible añadir al archivo *MainConfig.xml* los parámetros definidos para el nuevo canal con el fin de comprobar su correcto funcionamiento.