



INFRASTRUCTURE MANAGED JAVA KP (KP/APP MODEL)

June 2016

Version 2



1 INDEX

1	INDEX	2
2	INTRODUCTION	4
2.1	GOAL AND SCOPE OF THE CURRENT DOCUMENT	4
3	MANAGED JAVA KP'S	5
3.1	CONCEPTS	5
3.2	FUNCTIONAL LOGIC	5
3.3	KP LAUNCHER	6
3.3.1	Running the KP Launcher	7
3.4	APP MODEL (KP MODEL).....	8
3.4.1	Events.....	10
3.4.2	Workers.....	11
3.4.3	Subscription.....	17
3.4.4	JMX Monitoring.....	18
3.4.5	Example of Use.....	18
4	EXAMPLE OF FUNCTION: APP MODEL	20
4.1	STRUCTURE AND CONFIGURATION.....	20
4.2	LAUNCHING	23
4.3	COMPILING THE SOURCES	24
4.4	CLASS ORGANIZATION	25
4.4.1	StartApp and APPStop Workers	25
4.4.2	DataInputLoop Worker	26
4.4.3	PrepareMessageToSend Worker	26
4.4.4	SendServiceWrapper Worker.....	27
4.4.5	DataSendToSib and ErrorSendToSib Workers	28
4.4.6	SubscriptionListener Worker	28
4.4.7	Connection Worker	29

4.4.8	NoConnection Worker	29
4.4.9	Monitoring Worker	30
4.4.10	Business package classes	30
5	ANNEXES	31
5.1	INSTALLING SOFIA2-SDK	31



2 INTRODUCTION

2.1 Goal and scope of the current document

This document's objective is describing the KP Model (Managed Java KP's infrastructure) and explaining its functioning through an example of use.

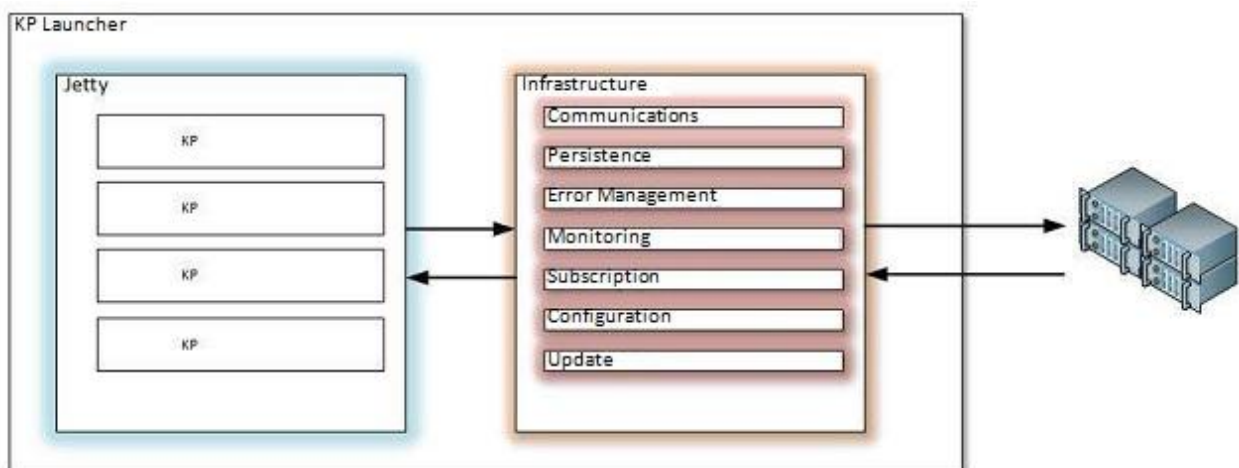


3 MANAGED JAVA KP's

3.1 Concepts

- **KP Launcher:** This is the execution platform for the Self-managed KP's. It includes an Embedded Jetty Web KP Launcher Server that executes the different AppModelos and manages their configuration and update following the requirements that are specified from the SIB.
- **AppModelo = KP Model:** These are the KP that were developed following the methodology and requirements specified by the KP Launcher. They abstract the developer from the logic used for updates, configuration and connectivity loss.
- **Event:** These are the different actions that a KP can perform (Sensory catching, information transformation, sending and receiving data to and from the SIB).
- **Worker:** Those are the tasks that intercept the different events generated by the KP's.

3.2 Functional logic



- The launcher connects to the SIB and recovers the information provided by the SIB with the list of the KP's that must be deployed and their configuration.
- Check whether the Configuration or SW version is the one that the SIB establishes.
- Make a backup copy of the Configuration and SW of the KP's that must be updated.
- Update the Configuration and/or SW of the KP's with a version different from the one that the SIB specifies, no matter if their version is higher or lower than it.
- The launcher starts each of the KPs that are configured.

- The KPs work autonomously, accessing the infrastructure that the KP Launcher manages and that provides them with capacities to communicate with the SIB, monitoring, persistence.
- The Launcher asks the SIB periodically for the configuration of the KP that it must have deployed. If the SIB notifies a new version of Software of Configuration, the application is stopped and updated.

3.3 KP Launcher

As said, the KP Launcher is in charge of executing and managing the KPs in the Sofia2 platform.

It establishes an execution framework with a strict life cycle that eases the development of clients (APPModelo). Its goal is allowing for the AppModelos to be remotely maintained, centralising their configuration and version management in the SIB, easing the development of clients by providing mechanisms to manage errors, persistence and monitoring.

The KP Launcher is a JAR that raises a web container where it deploys the AppModelos with the configuration that the SIB notifies. It is a container that allows to simultaneously running different implementations of a KP on the same virtual machine. It uses a model of Workers to offer a framework of basic functionalities that are common to SOFIA's KP's (connection tools, sending and receiving messages, subscriptions, monitoring, local persistence of messages, retrying to send a failed message, automatic KP update/versioning, ...).

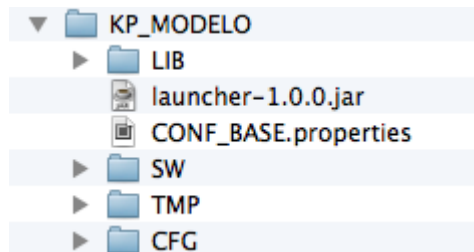
To work properly, it requires a basic configuration in the file CONF_BASE.properties

BASIC PROPERTIES OF THE KP LAUNCHER	
TIMEOUT	TimeOut for communications with the SIB
TIME_4_NEWVERSION	Specifies how often (in minutes) must check the configuration of the AppModelos that are deployed in the KPModelo.
SW_RUTA	Path where the AppModelo's WAR are stored.
CONF_RUTA	Path where the AppModelo's configuration properties is stored.
TMP_RUTA	Path where the backup copies are stored.
SIB	IP: MQTT port for communication with the SIB.

PORT	Port where the KPModelo's web server is raised.
KP	KP used to communicate with the SIB.
INSTANCIA_KP	KP instance used to communicate with the SIB
TOKEN	Token used to communicate with the SIB.

These are the minimum needed properties to run the KP Launcher, and they can be accessed from the AppModelos as long as the later does not overwrites them in its configuration file.

This is its structure:



- LIB where the dependencies to run KPModelo are stored.
- SW directory where the APPModelo's WAR are stored.
- CFG directory where each APPModelo's configuraiton is stored.
- TPM directory where the backup copy of the APPModelos to be updated is stored.

The KP Launcher must be run with the parameter `-DCONF_BASE= [DIRECTORY WHERE CONF_BASE.properties IS LOCATED]`

The script `START-KP-MODELO` runs the application.

If we don't run the KP Launcher with this configuration, it will look for the `CONF_BASE.properties` file in the `usr` directory.

3.3.1 Running the KP Launcher

When the KP Launcher is ran using the command `START-KP-MODELO`:

```
[~/Downloads/SOFIA2/Desarrollo] SOFIA2$ START-KP-MODELO
#####
                        RUNTIME SOFIA2
#####
.
. Starting KP-MODELO...
log4j:WARN No appenders could be found for logger (org.eclipse.jetty.util.log).
log4j:WARN Please initialize the log4j system properly.
log4j:WARN See http://logging.apache.org/log4j/1.2/faq.html#noconfig for more info.
```

The first thing it does is connecting to the SIB with the configuration specified in the file `CONF_BASE.properties`, requesting the configuration associated to it at the level of KP and KP instance.

The SIB will return on one side the global configuration of the KP:

- Ontologies configuration.
- Assets configuration.

The list of applications to be deployed, with their information at the level of

- Firmware
- Configuration

Each of these data is version. Thus, with its Firmware or Configuration's version is different (higher or lower) than the deployed one, then the KP Launcher will be updated. Before updating firmware or configuration, it stores a copy in the directory specified in the configuration parameter `TMP_RUTA`.

Once updated, the APPModelo is deployed in the KP Launcher.

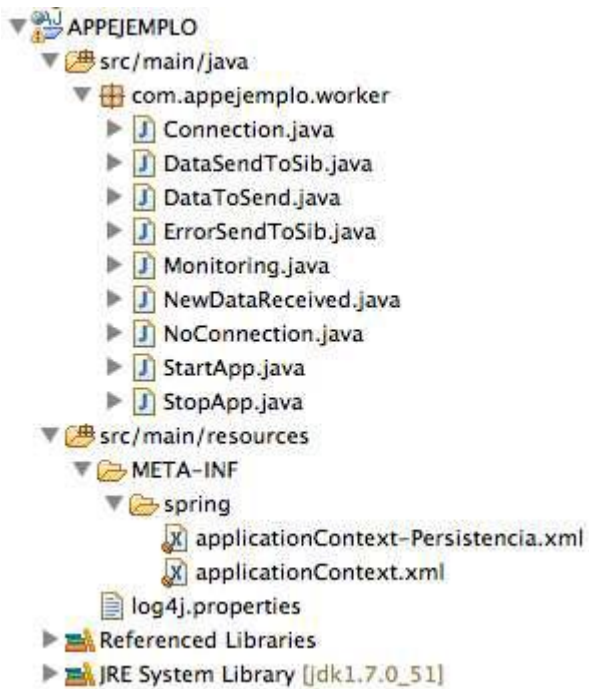
IMPORTANT

The applications already deployed in the KP Launcher and not notified by the SIB are deleted. If they are running, they will be stopped and then deleted.

3.4 APP Model (KP Model)

The applications that are deployed on the KP Lancher are called APPModelos. These applications are created with the command **`sofia2 crearAppModelo --id [APPMODELO'S NAME] --paquetebase [APPLICATION'S PACKAGE]`**. The Productivity Tool generates a structure and default implementations to start developing the specific business.





The generated Maven project has the needed dependencies to compile the AppModelos.

```
<dependency>
  <groupId>com.indra.sofia2</groupId>
  <artifactId>launcher</artifactId>
  <version>1.0</version>
  <scope>provided</scope>
</dependency>
<dependency>
  <groupId>com.mycila</groupId>
  <artifactId>mycila-pubsub</artifactId>
  <version>5.0.ga</version>
  <scope>provided</scope>
</dependency>
<dependency>
  <groupId>com.indra.sofia2</groupId>
  <artifactId>ssap</artifactId>
  <version>2.5.0</version>
  <scope>provided</scope>
</dependency>
<dependency>
  <groupId>com.jsontojava</groupId>
  <artifactId>jsontojava</artifactId>
  <version>1.0</version>
  <scope>provided</scope>
</dependency>
```

All these dependencies have scope provided because it is in run time that the KP Launcher provides with the versions of all the needed libraries to run the different APPModelos.

IMPORTANT

When multiple WARs are deployed, they must have the identifier as a context-param property in the web.xml, to unambiguously discriminate each application.

```
<context-param>
  <param-name>webAppRootKey</param-name>
  <param-value>NOMBREAPP MODELO</param-value>
</context-param>
```

```
</context-param>
```

From version 2.8.0 of the SDK, whenever a new AppModel is created, this property is created automatically.

3.4.1 Events

The Managed KP operate on the basis of Events. These Events are produced when an option is triggered.

The following events are pre-defined the KP Launcher's infrastructure.

- **START_APP**: This event is generated when the application starts.
- **STOP_APP**: This event is generated when the application is requested to stop.
- **APP_STOPED**: This event is generated when the application stops.
- **PREPARE_TO_RECEIVED**: This event is generated when the application is started and the Workers associated to the **START_APP** event have been launched.
- **NEW_DATA_RECEIVED**: This event is generated whenever data from a sensor is received.
- **DATA_TO_SEND**: This event is generated when the data received from a sensor is transformed into a SSAP message to be sent to the SIB.
- **CONNECTION_TO_SIB**: This event is generated when there is connectivity with the SIB.
- **NO_CONNECTION_TO_SIB**: This event is generated when connectivity with the SIB is lost.
- **DATA_SEND_TO_SIB**: This event is generated when an SSAP Message is sent to the SIB.
- **ERROR_ON_SENT_TO_SIB**: This event is generated when there is an error while sending an SSAP Message to the SIB.

The developer can also register new events in the following way:

The events must be registered through the KPWorkerCfg Bean. To register new events, we must get a reference to this object:

```
@Autowired
protected KPWorkerCfg kPWorkerCfg;
```

then invoke the method **public void** `addEvent(String event, Class message, Subscriber worker)`. We must report the Event to this method. The Event is simply a String identifying it, the object class that the Worker intercepting this Event receives, and the Worker itself, which is the implementation of the Subscriber interface, defining what we will do whenever this event is created.

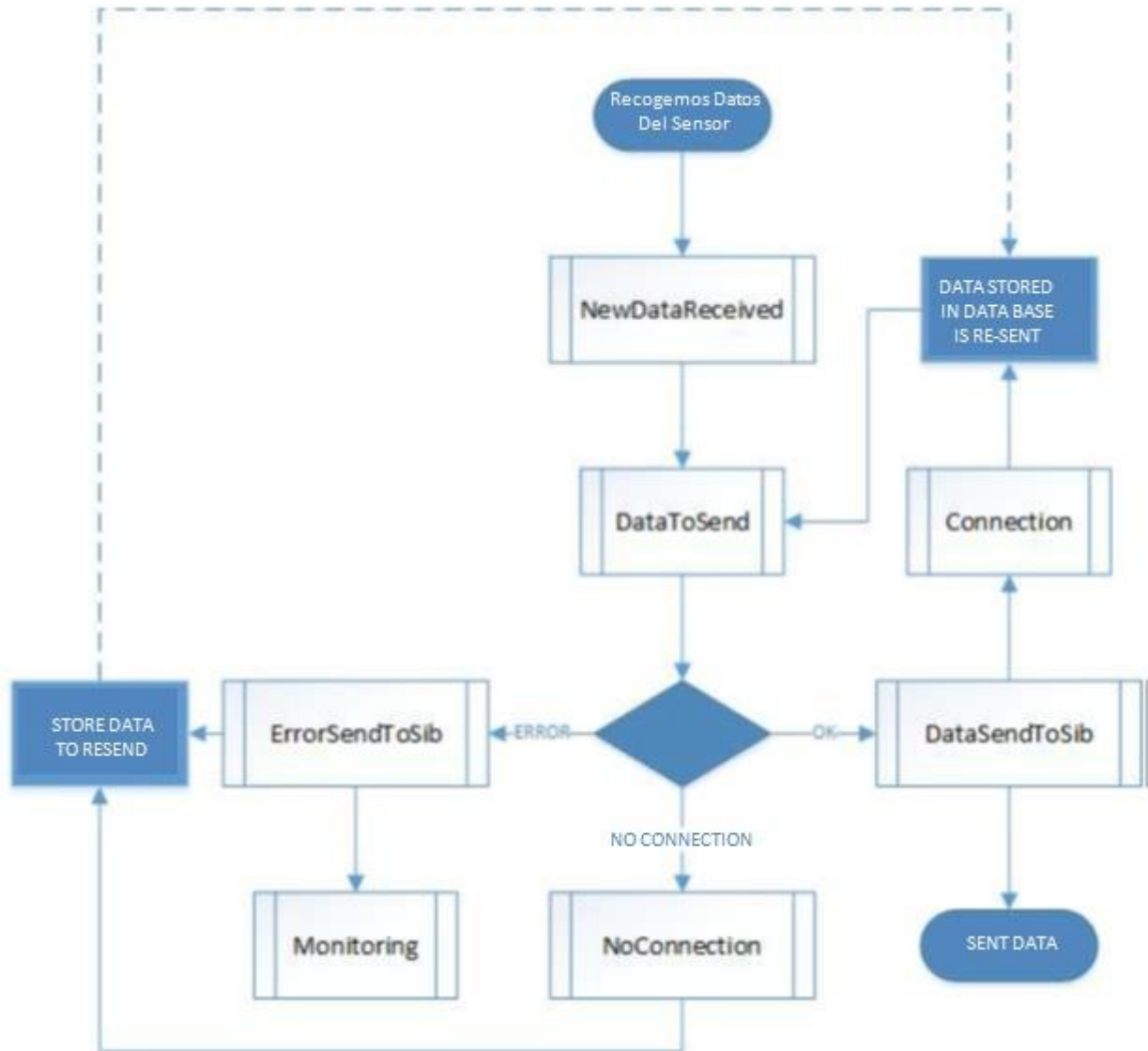
```
public class StartAppWorker extends Subscriber<LifeCicleMessage> {
    public void onEvent(Event<LifeCicleMessage> event) throws Exception{
```

For an implementation like the previous one, the invocation of the addEvent method will be like this:

```
kPWorkerCfg.addEvent("MI_EVENTO", LifeCicleMessage.class, new StartAppWorker())
```

3.4.2 Workers

The mode of operation is based on Workers using a Pub/Subscribe method at JVM level. These Workers define the following life cycle:



3.4.2.1 PrepareToReceived

```
@Component
public class PrepareToReceived extends PrepareToReceivedWorkerImpl {

    @Override
    public SensorMessage reveivedDataSensor (LifeCicleMessage lifeCicleMessage) {

        /*
        * METODO QUE ES EJECUTADO DE FORMA CICLICA ES EL ENCARGADO DE LEER LA
        INFORMACION
        * SENSORICA HA DE DEVOLVER UN OBJETO SensorMessage CON LA INFORMACIN DE LOS
        * SONSORES.
        */
    }
}
```

```

        */
        //TODO
        return new SensorMessage();
    }
}

```

This Worker is automatically executed when the APPModel is started. It runs a loop that constantly sends the information gathered in the method `receivedDataSensor` to the event `NEW_DATA_RECEIVED`.

3.4.2.2 NewDataReceived

```

@Component
public class NewDataReceived extends NewDataReceivedWorkerImpl {

    @Override
    public SSAPMessage generateSSAPMessage(SensorMessage sensorData) {
        /*
         * METODO QUE ES EJECUTADO CUANDO EL CAPTADOR DE DATOS SENSORICOS LOS NOTIFICA
         * ESTE METODO HA DE TRANSFORMAR LOS DATOS CAPTADOS DE LOS SENSORES EN UN SSAP
         * SI ESTE METODO DEVUELVE UN OBJETO DISTINTO DE NULO LO ENVIA AUTOMATICAMENTE A
         * DataToSend EL SESSION KEY NO ES NECESARIO PUES LA PLATAFORMA LO RELLENARA
         */
        //TODO
        return new SSAPMessage();
    }
}

```

It is the Worker we must invoke when we receive data from the sensor. Its goal is transforming the `SensorMessage` containing the information we received from the Sensors into a `SSAPMessage`.

When we return the `SSAPMessage`, the abstract class we extend from will take care of sending the message to the next Worker.

```

public void onEvent(Event<SensorMessage> event) throws Exception{
    SSAPMessage ssapMessage = generateSSAPMessage(event.getSource());
    if (ssapMessage!=null) {
        kpWorkerCfg.publish(SensorEvents.DATA_TO_SEND.name(), ssapMessage);
    }
}

```

IMPORTANT

For the sensor's data to reach the `NewDataReceived` Worker, we must notify it with the following code:

```
kpWorkerCfg.publish(SensorEvents.NEW_DATA_RECEIVED.name(), sensorMessage);
```

The `kpWorkerCfg` object we can recover from Spring's context. The easiest way is with an `Autowired` in a Spring Bean:

```

@Autowired
protected KPWorkerCfg kpWorkerCfg;

```

3.4.2.3 DataToSend

```
@Component
public class DataToSend extends DataToSendWorkerImpl {

    @Override
    public SSAPMessage preProcessSSAPMessage(SSAPMessage requestMessage) {
        /*
         * METODO QUE ES EJECUTADO JUSTO ANTES DE ENVIAR EL SSAP CREADO EN
         * NewDataReceived AL SIB
         * EL MENSAJE ENVIADO SERA EL QUE DEVUELVA ESTE METODO PARA ENVIAR EL MENSAJE
         * GENERADO PREVIAMENTE
         * DEVOLVER EL OBJETO DE ENTRADA requestMessage SIN MODIFICAR
         */
        //TODO
        return requestMessage;
    }

    @Override
    public void postProcessSSAPMessage(SSAPMessage requestMessage, SSAPMessage
responseMessage) {
        /*
         * METODO QUE ES EJECUTADO DESPUES DE ENVIAR EL SSAP AL SIB LOS PARAMETROS QUE
         * SON EL requestMessage
         * MENSAJE ENVIADO Y EL requestMessage MENSAJE DE RESPUESTA DEL SIB
         */
        //TODO
    }
}
```

It is the Worker in charge of sending the SSAPMessage to the SIB. It has two methods where we implement the code from pre- preProcessSSAPMessage and post- postProcessSSAPMessage -processing the message. The abstract class we extend from will take care of sending the message to the next Worker.

```
public void onEvent(Event<SSAPMessage> event) throws Exception{
    SSAPMessage requestMessage = preProcessSSAPMessage(event.getSource());
    SSAPMessage responseMessage = sib.send(requestMessage);
    if (requestMessage!=null) {
        postProcessSSAPMessage(requestMessage, responseMessage);
    }
}
```

3.4.2.4 DataSendToSib

```
@Component
public class DataSendToSib extends DataSendToSIBWorkerImpl {

    @Override
    public void send(SibMessage message) {
        /*
         * METODO QUE ES EJECUTADO CUANDO SE HAN ENVIADO DATOS AL SIB DE FORMA CORRECTA
         * SI EL MENSAJE
         * ENVIADO ESTA EN LA BASE DE DATOS DE ERRORES LO BORRA DE ESTA
         */
        //TODO
    }
}
```

This Worker is notified when a message to the SIB has been correctly sent. The method we must implement receives a SibMessage that has the sent message and the SIB's response. The abstract class we extend from will take care of checking whether the sent message is in the data base's fail table or not and, if it is, of deleting it.

```
public void onEvent(final Event<SibMessage> event) throws Exception{
    try{
```

```

        persistence.getTransactionManager().callInTransaction(new
Callable<Void>() {
    public Void call() throws Exception {
        Table table =
persistence.findById(event.getSource().getRequest().toJson());
        if (table!=null){
            persistence.delete(table);
        }
        return null;
    }
});
} catch (Exception e){
    e.printStackTrace();
}
sended(event.getSource());
}

```

3.4.2.5 Connection

```

@Override
public void connected(SibMessage connected) {
    /*
    * METODO QUE ES EJECUTADO CUANDO SE HA REALIZADO UNA CONEXION CON EL SIB PREVIO
    A ESTE METODO
    * LA CLASE COMPRUEBA SI EXISTEN SSAP NO ENVIADOS O CON ERRORES EN LA BASE DE
    DATOS LOS VUELVE
    * A ENVIAR Y LOS BORRA DE LA BASE DE DATOS
    */
    //TODO
}

```

This Worker is notified when there is connectivity with the SIB. The abstract class we extend from will take care of recovering all the messages that have failed when being sent, and of retrying to send them.

```

public void onEvent(final Event<SibMessage> event) throws Exception{
    try{
        persistence.getTransactionManager().callInTransaction(new
Callable<Void>() {
    public Void call() throws Exception {
        List<Table> tables = persistence.findAll();
        for (Table table : tables){

            kPWorkerCfg.publish(SensorEvents.DATA_TO_SEND.name(),
SSAPMessage.fromJsonToSSAPMessage(table.getSsapMessage()));
        }
        return null;
    }
});
} catch (Exception e){
    e.printStackTrace();
}
connected(event.getSource());
}

```

3.4.2.6 NoConnection

```

@Component
public class NoConnection extends NoConnectionToSIBWorkerImpl {

    @Override
    public void noConnected(ErrorMessage error) {
        /*
        * METODO QUE ES EJECUTADO CUANDO NO DE PUEDE CONECTAR CON EL SIB
        */
        //TODO
    }
}

```

This Worker is notified when there is no connectivity with the SIB. This event also causes ErrorSendToSib.

3.4.2.7 ErrorSendToSib

```
@Component
public class ErrorSendToSib extends ErrorSendToSIBWorkerImpl {

    @Override
    public MonitoringMessage toMonitoring(ErrorMessage error) {
        /*
         * METODO QUE ES EJECUTADO CUANDO SE HAN ENVIADO DATOS AL SIB Y SE PRODUCE UN
         * ERROR EN EL ENVIO
         * ESTE METODO TRANSFORMA EL MENSAJE ERRORMESSAGE error EN UN MENSAJE
         * MonitoringMessage QUE SERA
         * INTERCEPTADO POR LA CLASE Monitoring
         *
         * SI EL MENSAJE DEVUELTO ES DISTINTO DE NULL PUBLICA
         */
        publish(InfrastructureEvents.MONITORING.name(), monitoringMessage);
        /*
         * //TODO
         */
        return new MonitoringMessage(error);
    }

    @Override
    public void onError(ErrorMessage error) {
        /*
         * METODO QUE ES EJECUTADO CUANDO SE HAN ENVIADO DATOS AL SIB Y SE PRODUCE UN
         * ERROR EN EL ENVIO
         * ANTES DE EJECUTAR ESTE METODO DE FORMA AUTOMATICA SE ALMACENA EL SSAP ENVIADO
         * EN LA BASE DE DATOS
         * CONFIGURADA PARA SU POSTERIOR REENVIO
         *
         * ESTE METODO PODRIA A PARTE DE ESCRIBIR EL LOG ADECUADO ELIMINAR DE LA BASE DE
         * DATOS EL SSAP SI SE DETECTA
         * QUE LO HA PROVOCADO UN ERROR SEMANTICO O SINTACTICO
         */
        /*
         * //TODO
         */
    }
}
```

This Worker is notified when there is no error when sending an SSAP Message to the SIB. The methods we must implement receive ErrorMessage type parameters.

```
public class ErrorMessage extends Exception{

    /**
     *
     */
    private static final long serialVersionUID = -8835699096763715136L;

    private SSAPMessage ssapRequestMessage;
    private SSAPMessage ssapResponseMessage;
    private Throwable exception;
}
```

These objects can contain the message sent to the SIB, the SIB's response and the Exception.

IMPORTANT

Not all the attributes are required to be informed. If there is an error before preparing the message to be sent, we will only have the originated Exception.

The abstract class we extend from will take care of storing the SSAP Message in the data base to send it to the SIB later.

```
public void onEvent(final Event<ErrorMessage> event) throws Exception{
    persistence.getTransactionManager().callInTransaction(new Callable<Void>() {
```




```

        public Void call() throws Exception {
            Table table = new
Table(event.getSource().getSsapRequestMessage().toJson());
            persistence.create(table);
            return null;
        }
    });

    onError(event.getSource());
    MonitoringMessage monitoringMessage = toMonitoring(event.getSource());
    if (monitoringMessage!=null) {
        kPWorkerCfg.publish(InfrastructureEvents.MONITORING.name(),
monitoringMessage);
    }
}

```

3.4.2.8 StopApp

```

@Component
public class StopApp extends StopAppWorkerImpl {

    @Autowired
    private KPWorkerCfg cfg;
    @Autowired
    private PropertyPlaceholder property;

    @Override
    public void stopApp(LifeCicleMessage message) {
        /*
        * METODO QUE ES EJECUTADO CUANDO SE SOLICITA LA DETENCION DE LA APLICACION
        * EN ESTE METODO SE DEBERIA DE DETENER LA LECTURA SENSORICA Y REALIZAR LOS
        * ADECUADOS PARA DETENER DE FORMA SEGURA LA APLICACION
        */
        //TODO
    }
}

```

This Worker is notified by the KPModel when an application is requested to be stopped. In the method that we must implement, we must add the needed measures for the imminent stop of the application. The abstract class we extend from will take care of notifying the KPModel when it can stop the application.

```

public void onEvent (final Event<ErrorMessage> event) throws Exception{
    persistence.getTransactionManager().callInTransaction(new Callable<Void>() {
        public Void call() throws Exception {
            Table table = new
Table(event.getSource().getSsapRequestMessage().toJson());
            persistence.create(table);
            return null;
        }
    });

    onError(event.getSource());
    MonitoringMessage monitoringMessage = toMonitoring(event.getSource());
    if (monitoringMessage!=null) {
        kPWorkerCfg.publish(InfrastructureEvents.MONITORING.name(),
monitoringMessage);
    }
}

```

IMPORTANT

The other Classes can be implemented without inheriting from the abstract class by using interfaces. This Worker must mandatorily implement a class that inherits from



ErrorSendToSibWorkerImpl. If it does not exist, the KPMModelo's default implementation will be used; it notifies directly the KPMModel on the possibility to stop the application.

3.4.3 Subscription

Subscription to events notified by the SIB is done by encapsulating the Listener mechanism provided by SOFIA2 Java API.

The subscribers will be in charge of acting upon the notifications to which they are associated.

To create a subscriber, we must extend the class `com.indra.sofia2.kpmodelo.infraestructure.subscription.Subscriber`.

```
@Component
public class SubscriptionListener extends Subscriber {

    @Override
    @PostConstruct
    public void init() throws SubscriptionException {
        /*
         * METODO EN EL QUE ESTABLECEMOS QUE SUSCRIPCION ATENDERA ESTE LISTENER

        subscribe(ontology, query, SSAPQueryType);

        */
        //TODO
    }

    @Override
    public void onEvent(SSAPMessage arg0) {
        /*
         * METODO QUE ES EJECUTADO CUANDO SE NOTIFICA LA INFORMACION A LA QUE NOS HEMOS
         * SUSCRITO EN EL INIT
         * DE LA CLASE
        */
        //TODO
    }
}
```

We will define the class as a Spring bean by noting it with `@Component` and the init method will be noted with `@PostConstruct`. Thus we will ensure that it will be executed when the Bean is registered.

By extending the Subscriber class, we will have the following methods available:

```
void subscribe(String ontology, String query, SSAPQueryType queryType) throws SubscriptionException

void unsubscribe() throws SubscriptionException
```

which allow us to subscribe to an ontology and to cancel that subscription. When the SIB notifies the information about our subscription, the method

```
void onEvent(SSAPMessage arg0)
```

will receive the SSAP message sent by the SIB, and we will be able to use it. Additionally, we can get the information about the subscription with the method

```
SubscriptionData getSubscriptionData()
```

which has the following information structure:

```
private String subscriptionId;
```

```
private String ontology;
private SSAPQueryType queryType;
private String query;
```

3.4.4 JMX Monitoring

Same as with Subscription, Monitoring through JMX requires the MBeans to register in the MBeans server. To do this, the platform identifies all the Beans that implement the interface `JmxSelfNaming` and registers them as MBeans to allow for the exploitation of their information. We must note the Beans with the Mycila JMX notes in charge of registering the information following our parameterization in the MBean server.

<http://code.mycila.com/jmx/#documentationXXX>

```
@JmxBean("com.company:type=MyService,name=main")
public final class MyService {

    private String name;

    @JmxField
    private int internalField = 10;

    @JmxProperty
    public String getName() { return name; }

    public void setName(String name) { this.name = name; }

    @JmxMethod(parameters = {@JmxParam(value = "number", description = "put a big number please
    !")})
    void increment(int n) {
        internalField += n;
    }
}
```

3.4.5 Example of Use

The command **>sofia2 crearAppModelo** creates the AppModelo application structure with the basic configuration and the predefined Workers for the developers to modify it.

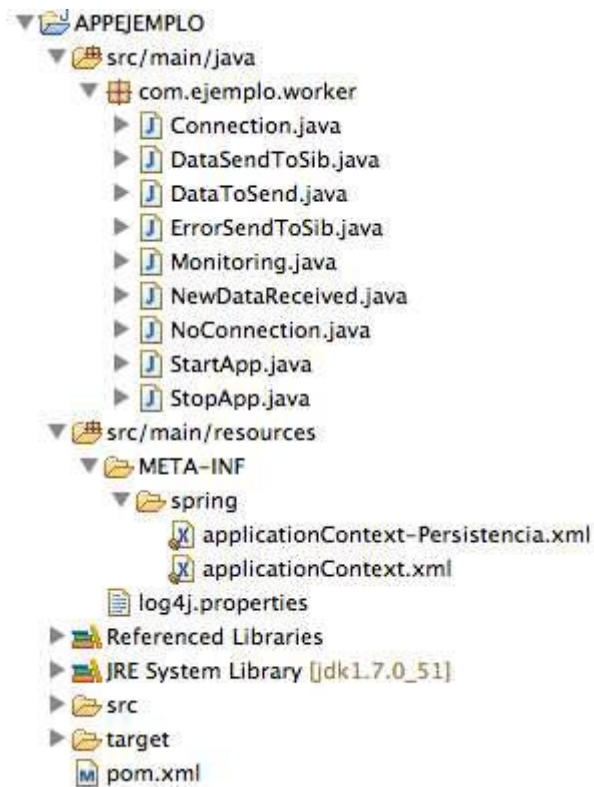
>sofia2 crearAppModelo --id appEjemplo --paquetebase com.ejemplo

```
sofia2> sofia2 crearAppModelo --id appEjemplo --paquetebase com.ejemplo
*****
SE INICIA LA CREACION DE LA APPMODELO appEjemplo
*****

[INFO] Scanning for projects...
sofia2> Se ha creado la estructura para la APPMODELO APPEJEMPLO
*****PROCESO FINALIZADO*****

[INFO]
[INFO]
[INFO] Building app 1.0
[INFO]
[WARNING] The POM for org.springframework.roo:org.springframework.roo.annotations:jar:1.2.4.RELEASE is missing, no dependency information available
[INFO]
[INFO] --- maven-resources-plugin:2.5:resources (default-resources) @ APPEJEMPLO ---
[debug] execute contextualize
[WARNING] Using platform encoding (UTF-8 actually) to copy filtered resources, i.e. build is platform dependent!
[INFO] Copying 3 resources
[INFO]
[INFO] --- maven-compiler-plugin:2.3.2:compile (default-compile) @ APPEJEMPLO ---
[WARNING] File encoding has not been set, using platform encoding UTF-8, i.e. build is platform dependent!
[INFO] Compiling 9 source files to /Users/pedrotorihioquardisola/Downloads/SOFIA2/pr/target/classes
[INFO]
[INFO] --- maven-resources-plugin:2.5:testResources (default-testResources) @ APPEJEMPLO ---
[debug] execute contextualize
[WARNING] Using platform encoding (UTF-8 actually) to copy filtered resources, i.e. build is platform dependent!
```

which generates the project directly importable in Eclipse.



IMPORTANT

In the file `applicationContext.xml`

```
<bean id="propertyPlaceholder"
class="com.indra.sofia2.kpmodelo.infraestructure.loader.PropertyPlaceholder">
    <constructor-arg value="APPEJEMPLO"/>
</bean>
```

The bean `propertyPlaceholder` specifies the application identifier. This value cannot be modified because it is the contract between the `KPModelo` and `AppModelo`.

The WAR's name and the configuration file must be named with the value of this identifier.

Remember that the application's identifier is needed in the file `web.xml` as a `context-param` property when several applications are deployed.

4 EXAMPLE OF FUNCTION: APP Model

Aiming to clarify concepts that the previous section explained, we include a complete example of Managed Java KP Infrastructure (KP/AppModelo).

Before going on with this guide, we recommend reading the guide **SOFIA2-SOFIA2 Concepts.pdf** and install Sofia2's SDK as explained in the current document's annex.

The described example can be downloaded from this URL:

<http://sofia2.org/owncloud/public.php?service=files&t=956a786e3ac0a9cb804d185c6a70aa7a>

The **Simplestat** App is an AppModelo used in a real gateway's functional test. It is a hub of sensory signals running on Java and a Debian Wheezy operating system.

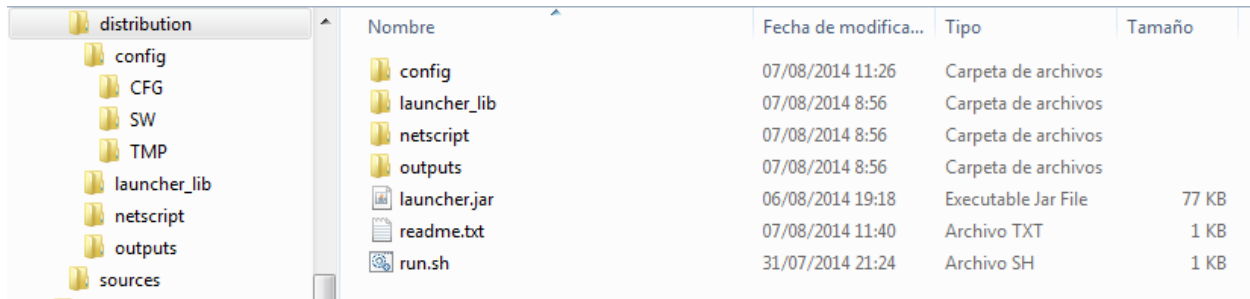
The test measures a KP's publication and subscription capacities, response time, possible message loss, etc.

It focuses in testing some of the capabilities of the KP Model:

- Simulated reading of data from a GPIO.
- The KPModelo's publishing/subscribing capabilities.
- Saving messages in case of error.
- Reconnection and re-sending capabilities.
- Leaving a log of all the operations, plus the OS's status (memory, CPU).

4.1 Structure and configuration

This is the example distribution's file structure.



Nombre	Fecha de modifica...	Tipo	Tamaño
config	07/08/2014 11:26	Carpeta de archivos	
launcher_lib	07/08/2014 8:56	Carpeta de archivos	
netscript	07/08/2014 8:56	Carpeta de archivos	
outputs	07/08/2014 8:56	Carpeta de archivos	
launcher.jar	06/08/2014 19:18	Executable Jar File	77 KB
readme.txt	07/08/2014 11:40	Archivo TXT	1 KB
run.sh	31/07/2014 21:24	Archivo SH	1 KB

It will work as is on the objective system, as long as the configuration files and Java paths coincide.

The root includes the KPModelo's functioning main jar and a run-sh script to launch it.

The **config** folder has the KPMModelo's general configurations.

distribution

config

CFG

SW

TMP

launcher_lib

netscript

outputs

Nombre	Fecha de modifica...	Tipo	Tamaño
CFG	07/08/2014 8:56	Carpeta de archivos	
SW	07/08/2014 8:56	Carpeta de archivos	
TMP	07/08/2014 8:56	Carpeta de archivos	
CONF_BASE.properties	31/07/2014 21:04	Archivo PROPERTI...	1 KB
CONF_MODULO.properties	31/07/2014 15:26	Archivo PROPERTI...	1 KB

The files **CONF_BASE** and **CONF_MODULO** must be placed as described. We need to specify correctly the path and point to the SIB, port, token and default KP correctly. The default data base is in memory and requires no modification:

The CONF folder has the configuration for the APPS contained in the KPMModel. In this case the name must be CONF_ application's name:

distribution

config

CFG

Nombre	Fecha de modifica...	Tipo	Tamaño
<div><div></div>CONF_SIMPLESTAT.properties</div>	31/07/2014 15:47	Archivo PROPERTI...	1 KB

Within it, we have persisted the data base to disk to avoid memory saturation. We specify the path:

```

CONF_SIMPLESTAT.properties
1 IDAPP=SIMPLESTAT
2 SIB=
3 KP=
4 INSTANCIA_KP=
5 TOKEN=
6 LAST_VERSION_SW_OK=1
7 LAST_VERSION_CFG_OK=1
8 #CONFIGURACION DE BASE DE DATOS
9 database.url=jdbc:h2:file:/home/gcsc/kpgenerico_subs/config/TMP/simplestat;DB_CLOSE_DELAY=-1
10 database.username=sa
11 database.password=
12

```

Creation is automatic. We can see a track of an empty data base from previous executions in the specified temporary directory:

Archivo

Edición

Ver

Herramientas

Ayuda

Organizar

Incluir en biblioteca

Compartir con

Grabar

Nueva carpeta


CFG

SW

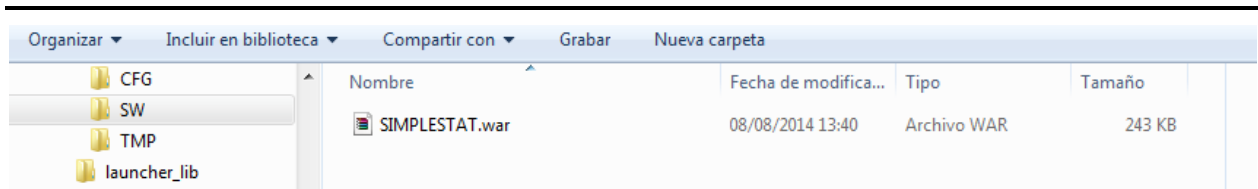
TMP

launcher_lib

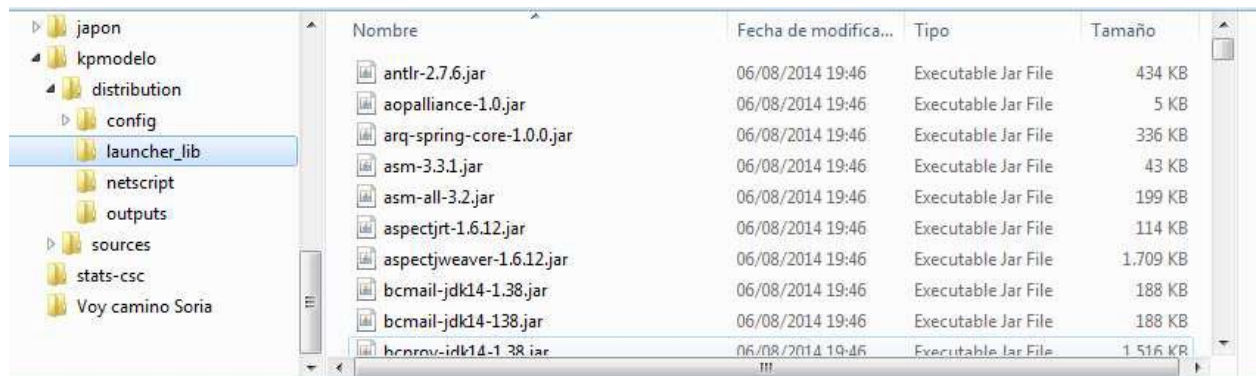
netscript

Nombre	Fecha de modifica...	Tipo	Tamaño
 simplestat.h2.db	06/08/2014 21:35	Data Base File	28 KB

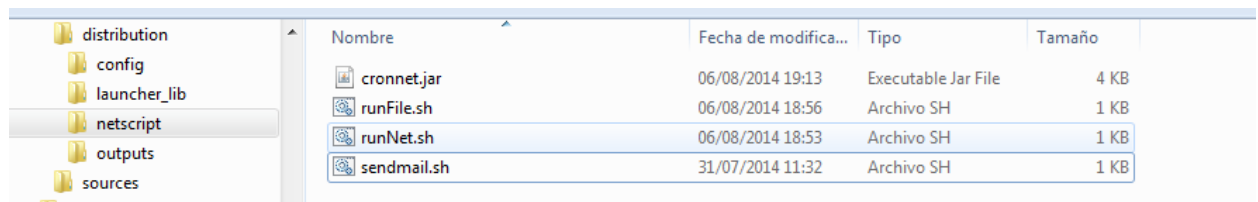
The folder SW includes, manually or through server's download, the wars for the APPS that will be deployed in the KpModelo:



The libraries must be placed within the reach of the Java classpath. In the case of the launching script run.sh, they must be in the distribution's root:

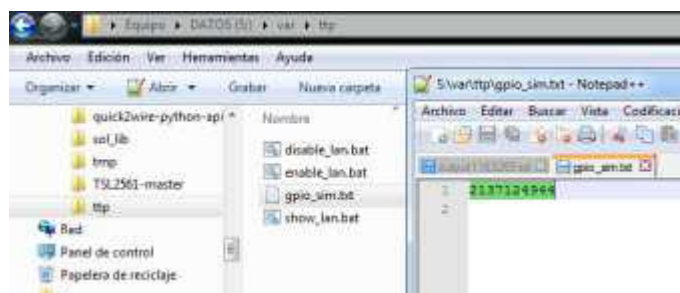


The netscript folder includes two utilities:



The script runFile.sh randomly modifies the content of the file /var/ttp/gpio_sim.txt (The file must be created).

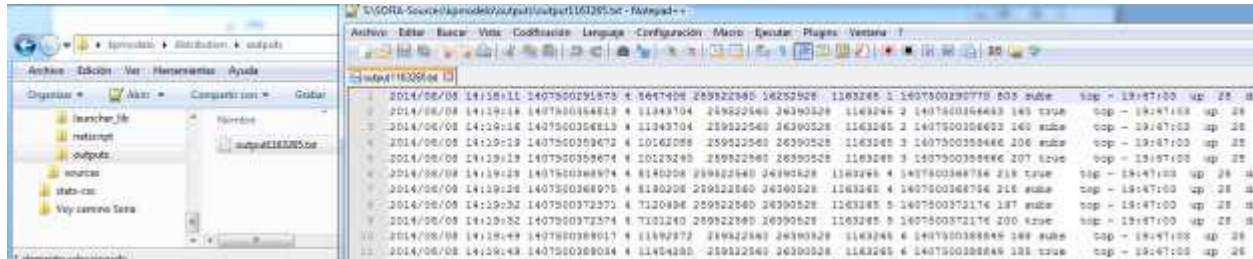
The goal is for the APP to use it as the source of simulated data from a GPIO sensor.



The script runNet.sh changes after a random time (default 5 minutes) between active and inactive network services.

Within each cycle, we can configure the file sendmail.sh to send information to an address with the activity times. This is interesting because we will lose access to the Gateway during a disconnection.

Finally, in the outputs folder we will write the metrics and make the study's statistics:



4.2 Launching

This is done from the script run.sh in the root. This script includes:

```
sudo echo 3 > /proc/sys/vm/drop_caches
sudo rm ./config/TMP/*
java -Xmx192m -XX:MaxPermSize=24m -DCONF_BASE=./config -classpath ./launcher_lib/*:./launcher.jar com.indra.sofia2.kpmodelo.KpModelo
```

echo 3>/proc/sys/vm/drop_caches	Cleans OS's caches to offer similar metrics in every test
rm ./config/TMP/*	Cleans temporaries
-Xmx192m	Cleans the stack. The gateway has a 256m limit
-XX:MaxPermSize=24m	perm size
-DCONF_BASE=./config	Property of the jvm environment specifying where the configuration folder is
-classpath	Each of the locations in the Java libraries, semicolon as separator (in Linux systems)
com.indra.sofia2.kpmodelo.KpModelo	Launcher's Java main class (with a main)

We recommend sending the script from its own folder as

```
>sudo nohup ./run.sh &
```

The scripts in the utility folder ./netscript will work correctly with the default options, but it is easy to edit and change them. We recommend to launch them from its folder in the following way:

>sudo ./runFile.sh &	Without console output. This script is needed for the APP to send data (It will send nothing if the content of the reading of simulated GPIO data is unchanged).
>sudo nohup ./runNet.sh &	Remember configuring the SO's mail command and pointing the right address in the script netscript/sendmail.sh if we want

Example of a correct trace:

4.3 Compiling the sources

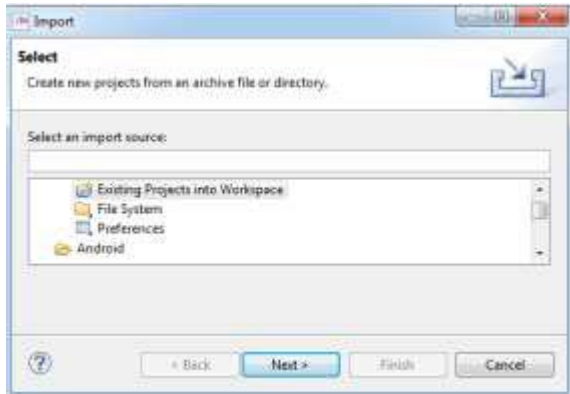
The Maven pom is provided to download and update the classes. Access to the Sofia public repository <http://sofia2.org/nexus/content/groups/public/> is required to update Sofia2's dependencies.

Compilation of the sources will generate in the target directory the APP Modelo's war. Rename and move the war to the following location so that it is incorporated to the KpModelo's container.



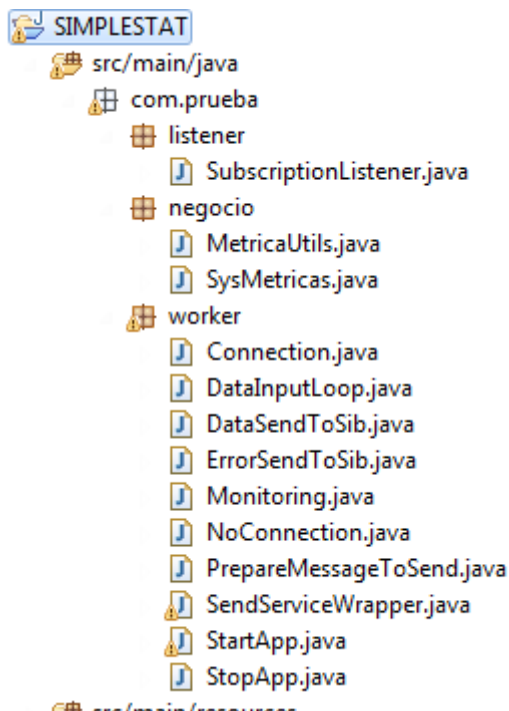

```
>mvn eclipse:eclipse
```

that allows to import the project to Eclipse.



4.4 Class organization

This is the structure of the SIMPLESTAT project:



Except for the negocio (Spanish for "business") folder, the whole class structure represent different Workers which will perform a task throughout the APP Modelo's life cycle. Workers have been created for informative purposes for each task type, but not all of the implement a functionality.

4.4.1 StartApp and APPStop Workers

It is executed when the APP Model is deployed or withdrawn (for updates). In this specific APP it has no functionality.

```
@Override
public void startApp(LifeCicleMessage arg0) {
    /*
     * METODO QUE ES EJECUTADO CUANDO SE PRODUCE EL INICIO DE LA APLICACION MODELO
     */
    Log.info( "My App started Now" );
}
```

4.4.2 DataInputLoop Worker

Extends from DataSendToSIBWorkerImpl. Periodical cycle to read sensor data. We need to mark the reading times (200ms) and to evaluate whether the data has changed (file date) to send it to the SIB as a SensorData object (if the method sends null, then the object is not sent).

```
@Override
public SensorMessage readDataSensor(LifeCicleMessage lifeCicleMessage) {
    /*
     * METODO QUE ES EJECUTADO DE FORMA CICLICA ES EL ENCARGADO DE LEER LA INFORMACIÓN SENSÓRICA HA DE DEVOLVER
     * UN OBJETO SensorMessage CON LA INFORMACIÓN DE LOS SENSORES
     */
    try {
        Thread.sleep( 200 );

        // lectura del fichero gpio_sim.txt
        FileReadResult readFileAsString = metricaUtils.readFileAsString( "/var/ttp/gpio_sim.txt" );

        // si ha cambiado la fecha del fichero desencadena mensaje y estadística
        if ( readFileAsString.date > lastDate ) {
            lastDate = readFileAsString.date;

            // compone SensorMessage (key, value)
            SensorMessage sensor = new SensorMessage();
            sensor.setProperty( "id", metricaUtils.nextLong() );
            sensor.setProperty( "data", readFileAsString.data );
            sensor.setProperty( "date", readFileAsString.date );

            Log.info( "1 DataInputLoop:readDataSensor" );

            // devolver para enviar al SIB
            return sensor;
        }
    } catch (InterruptedException e) {
        e.printStackTrace();
    }
}
```

4.4.3 PrepareMessageToSend Worker

Extends from NewDataReceivedWorkerImpl. Transforms the SensorData data properties object into a SSAP protocol INSERT message. In the example's specific case, a json template is read and the key values are replaced.

```

@Override
public SSAPMessage generateSSAPMessage(SensorMessage sensorData) {
    /*
     * METODO QUE ES EJECUTADO CUANDO EL CAPTADOR DE DATOS SENSORICOS LOS NOTIFICA
     * ESTE METODO HA DE TRANSFORMAR LOS DATOS CAPTADOS DE LOS SENSORES EN UN SSAP VALIDO
     * SI ESTE METODO DEVUELVE UN OBJETO DISTINTO DE NULO LO ENVIA AUTOMATICAMENTE A LA CLASE
     * DataToSend EL SESSION KEY NO ES NECESARIO PUES LA PLATAFORMA LO RELLENARA AUTOMATICAMENTE
     */
    Log.info( "3.1 PrepareMessageToSend:generateSSAPMessage" );

    // lerr plantilla json del mensaje a enviar
    if ( plantillMsg == null ) {
        plantillMsg = metricaUtils.readResourceAsString( "/dummymsg2.txt" );
    }

    // sustituir los valores de la cadena json
    String message = plantillMsg.replaceFirst( "<id>", "" + sensorData.getProperty( "id" ) );
    message = message.replaceFirst( "<description>", (String)sensorData.getProperty( "data" ) );
    message = message.replaceFirst( "<timesend>", "" + System.currentTimeMillis() );
    message = message.replaceAll( "\\n", "" );
    message = message.replaceAll( "\\r", "" );
    message = message.replaceAll( " ", "" );

    // generar mensaje SSAP y devolverlo listo para envio
    SSAPMessage ssap = new SSAPMessageGenerator().generateInsertMessage( null, "Stat", message );

    return ssap;
}

```

4.4.4 SendServiceWrapper Worker

Extends from X. Made up of two parts: The preProcessSSAPMessage method , that takes the message before sending it:

```

@Override
public SSAPMessage preProcessSSAPMessage(SSAPMessage requestMessage) {
    /*
     * METODO QUE ES EJECUTADO JUSTO ANTES DE ENVIAR EL SSAP CREADO EN NewDataReceived AL SIB
     * EL MENSAJE ENVIADO SERA EL QUE DEVUELVA ESTE METODO PARA ENVIAR EL MENSAJE GENERADO PREVIAMENTE
     * DEVOLVER EL OBJETO DE ENTRADA requestMessage SIN MODIFICAR
     */

    Log.info( "6 DataToSend:preProcessSSAPMessage" );

    // Añadir el mensaje al mapa de enviados
    String[] split = requestMessage.getBody().split( "id\\\\\\\\":\\\\\\\\" " ) [ 1 ].split( "\\\\\\\\" );
    long id = Long.parseLong( split[ 0 ] );
    metricaUtils.messagesMap.put( id, requestMessage );

    return requestMessage;
}

```

In this method we add the message to a map to check later whether the same message is returned as a subscription.

The postProcessMessage method takes the message after it has been sent, along with a message with the SIB's response include the status of the sending operation.

Metrics are written with the message's response times and whether it was correctly sent or not.

Not implemented Workers. DataSendToSIB is executed whenever the SIB sends a message correctly.

ErrorSentToSIB is executed whenever there is an error with a message (This is automatically stored in the data base for further re-tries).

Extends `Suscriber`. Bean of subscription to APPModelo's ontology. The subscription process is performed in two parts. First an initialization where we subscribe to an ontology query; in this case it is the same ontology where the messages are published:

The goal is receiving via subscription the same message we have published.

Then a listener for incoming messages, which also processes them:

```
@Override
public void onEvent(SSAPMessage arg0) {
    /*
     * METODO QUE ES EJECUTADO CUANDO SE NOTIFICA LA INFORMACION A LA QUE
     * NOS HEMOS SUSCRITO EN EL INIT DE LA CLASE
     */
    Log.info( "SubscriptionListenerSubscriptionListener:onEvent " );

    // parsear mensaje
    String[] split = arg0.getBody().split( "\\\"id\\\" : \\\"\\\" " );
    long id = Long.parseLong( split[ 0 ] );
    long timeSent = Long.parseLong( split[ 5 ].split( ":" )[ 1 ].split( "," )[ 0 ].trim() );

    // Quitar el mensaje del mapa de enviados
    metricaUtils.messagesMap.remove( id );

    // mensaje a fichero de estadísticas
    metricaUtils.appendMetrica( "" + id, "" + timeSent, "" + ( System.currentTimeMillis() - timeSent ), "subs" );
}
```

In the example's APP, we get the message's id then we delete the message from the published messages map, ensuring that the corresponding subscription has been received.

4.4.7 Connection Worker

Not implemented Worker. The connected method is executed whenever network connection is recovered.

```
@Override
public void connected(SibMessage connected) {
    /*
     * METODO QUE ES EJECUTADO CUANDO SE HA REALIZADO UNA CONEXION CON EL SIB PREVIO A ESTE METODO
     * LA CLASE COMPRUEBA SI EXISTEN SSAP NO ENVIADOS O CON ERRORES EN LA BASE DE DATOS LOS VUELVE
     * A ENVIAR Y LOS BORRA DE LA BASE DE DATOS
     */
    Log.debug( "Connection:connected" );
}
```

4.4.8 NoConnection Worker

Extends X. The noConnected method is executed whenever network connection is lost. In these cases, the APP checks whether the message storage map has more than one stored message or not; if it has, it sends an error to the metrics file.

As the connection can be lost between sending and subscription (Due to the test model, it is almost impossible to have concurrent messages because readings are not concurrent), an error is accepted.

```

public void noConnected(ErrorMessage error) {
    /*
     * METODO QUE ES EJECUTADO CUANDO NO SE PUEDE CONECTAR CON EL SIB
     */
    Log.info( "NoConnection:noConnected" );

    // en caso de desconexion chequear tamaño del mapa de mensajes a chequear
    // permitimos un error si justo sucedio la desconexion entre un envio-recepcion de suscriptor
    if ( metricaUtils.messagesMap.size() > 1 ) {
        Log.error( "Excesivo numero de mensajes no suscritos" + metricaUtils.messagesMap.size() );

        synchronized ( metricaUtils.messagesMap ) {
            Iterator<Entry<Long, SSAPMessage>> i = metricaUtils.messagesMap.entrySet().iterator();
            // Display elements
            while( i.hasNext() ) {
                Map.Entry<Long, SSAPMessage> me = i.next();
                SSAPMessage arg0 = me.getValue();

                String[] split = arg0.getBody().split( "\\\"id\\\" : \\\"\\\" " ) [ 1 ].split( "\\\" " );
                long id = Long.parseLong( split[ 0 ] );
                long timeSent = Long.parseLong( split[ 5 ].split( ":" ) [ 1 ].split( "," ) [ 0 ].trim() );
                Long.parseLong( split[ 5 ].split( ":" ) [ 1 ].split( "," ) [ 0 ] );

                metricaUtils.appendMetrica( "" + id, "" + timeSent, "" + ( System.currentTimeMillis() - timeSent ), "ERR" );
            }
        }
    }
    // limpiar mensajes
    metricaUtils.messagesMap.clear();
}

```

The map is then cleaned.

4.4.9 Monitoring Worker

Not implemented Worker. Used to monitor in case of errors and tracking.

4.4.10 Business package classes

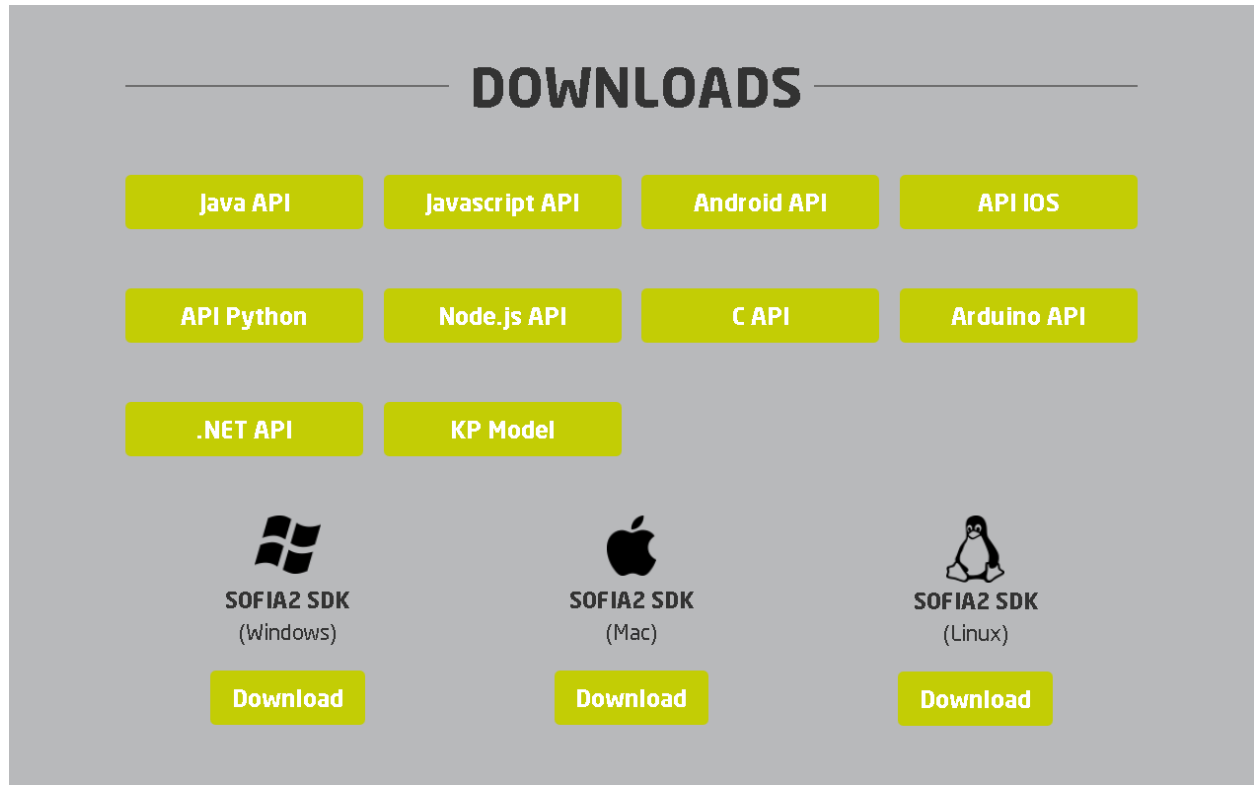
These classes are related to metrics collecting, such as writing in the information file, launching commands from the operating system (top), etc.

5 ANNEXES





5.1 Installing SOFIA2-SDK

To develop SOFIA2 applications, the SOFIA2 SDK must be installed. To do so:

Access http://sofia2.com/desarrollador_en.html and go to the Download sections. Download the SOFIA2 SDK.



It is a compressed file. Once uncompressed, it will create the following folder structure:

Nombre	Fecha de modifica...	Tipo	Tamaño
 SOFIA-RUNTIME	11/11/2013 10:25	Carpeta de archivos	
 SOFIA-SDK	11/11/2013 10:31	Carpeta de archivos	
 SOFIA-START.bat	15/07/2013 11:56	Archivo por lotes ...	1 KB
 SOFIA-STOP.bat	09/07/2013 16:38	Archivo por lotes ...	1 KB

The SOFIA-RUNTIME directory includes a set of tools to develop KP's (JDK, browser) along with a Java KP and a JavaScript KP as examples.

The SOFIA-SDK directory has a set of tools to develop KP's (IDE's, API's, Maven Repository with dependency libraries for Java Projects...).