# GUÍA DE USO

# MOTOR SCRIPTING

# SOFIA2

MAYO 2016

Version 5

indra

# 1 ÍNDICE

# 2 INTRODUCCIÓN

## 2.1 Requisito

Antes de seguir esta guía se recomienda leer la guía **SOFIA2-Conceptos SOFIA2.doc.**

## 2.2 Objetivos y alcance del presente documento

El presente documento describe cómo se usa el motor de scripting de SOFIA2 que permite a los usuarios de la plataforma crear sus propios scripts que se ejecutarán antes la llegada de una instancia de ontología o bien cada cierto tiempo.

## 2.3 Conceptos

- **Tipos de scripts:** la plataforma soporta 2 tipos de scripts:
  - **Scripts ante eventos:** permite ejecutar código ante llegada de instancias de ontologías
  - **Scripts temporizados:** permite ejecutar código cada cierto tiempo, este tiempo puede definirse con un cron.
- **Lenguaje de scripting:** los scripts se definen en lenguaje Groovy (**http://groovy.codehaus.org**).
- **APIs disponibles para los scripts (Operaciones):** los scripts pueden acceder a un conjunto de APIs que permiten ejecutar acciones como:
  - Enviar Mail.
  - Invocar URL.
  - Insertar instancia de Ontología.
  - Consultar instancias de una Ontología.
  - Borrar instancia de Ontología.

  Los usuarios con permisos podrán crear sus propias APIs.
- **Seguridad en los scripts:** los scripts se ejecutan en un contexto seguro, lo que garantiza que un error en un script afecte al resto, además tienen un tiempo de timeout en su invocación. Los scripts sólo pueden invocar a un conjunto de operaciones.

# 3 DESARROLLO DE SCRIPTS

## 3.1 Groovy

Los scripts se desarrollan en lenguaje Groovy, que permite programar en un lenguaje sencillo y no tipado.

Groovy se compila en ejecución a Java y es ejecutado en la JVM, y desde él se puede acceder a todas las clases y librerías Java.

Puede obtener más información sobre Groovy en **http://groovy.codehaus.org**.

## 3.2 API Operaciones

Las **Operaciones** permiten definir las APIs accesibles a los scripts, son los comandos que podrán ejecutar nuestros Script Groovy.

Las operaciones pueden definirse sólo por los usuarios administradores.

Un ejemplo de API es esta:

```
import com.indra.jee.arq.spring.sofia2.script.api.Api;

public class ApiGroovy {

	public class ScriptException extends Throwable{

		public ScriptException(Exception e){
			super(e);
		}

		public ScriptException(Exception e, String message){
			super(message, e);
		}

		public ScriptException(String message){
			super(message);
		}
	}

	private Api api;

	public ApiGroovy(){
		api = new Api();
	}

	public String insert(String ontologyName, String ontology) throws ScriptException{
		try{
			return api.insert(ontologyName, ontology);
		}catch(Exception e){
			throw new ScriptException(e);
		}
	}

	public String rolback(String ontologyName, String id) throws ScriptException{
		try{
			return api.rolback(ontologyName, id);
		}catch(Exception e){
			throw new ScriptException(e);
		}
	}
```

Este API se distribuye con la plataforma SOFIA2:

- No tiene definido ningún paquete, puesto que los Script no pueden importar Clases, nuestras operaciones deberán alojarse en el paquete base.
- Realiza la importación de una Clase, las operaciones no tienen restricciones y al usar Groovy se integra 100% con java.

```
import com.indra.jee.arq.spring.sofia2.script.api.Api;
```

- Pueden definir clases y subclases y deben definir métodos que expongan las operaciones que queremos usar.

```
public class ApiGroovy {
	public class ScriptException extends Throwable{
		public ScriptException(Exception e){
			super(e);
		}
```

### 3.2.1 ApiGroovy

La operación `ApiGroovy` operation is a facade to the Java API class offering those methos:

- **Insert** inserta una instancia de una ontología en la base de datos en tiempo real.
- **Rolbak** elimina una ontología de la base de datos en tiempo real.
- **getAtribute** obtiene el valor de un atributo de una ontología.
- **sendMail** permite enviar un email.

```
import com.indra.jee.arq.spring.sofia2.script.api.Api;
public class ApiGroovy {
	public class ScriptException extends Throwable{
		public ScriptException(Exception e){
			super(e);
		}
		public ScriptException(Exception e, String message){
			super(message, e);
		}
		public ScriptException(String message){
			super(message);
		}
	}
	private Api api;
	public ApiGroovy(){
		api = new Api();
	}
	public String insert(String ontologyName, String ontology) throws
ScriptException{
		try{
			return api.insert(ontologyName, ontology);
		}catch(Exception e){
			throw new ScriptException(e);
		}
	}
	public String rolback(String ontologyName, String id) throws
ScriptException{
		try{
```

```
                return api.rolback(ontologyName, id);
            }catch(Exception e){
                throw new ScriptException(e);
            }
      }
      public String getAtribute(String ontology, String atribute) throws
ScriptException{
            try{
                return api.getAtribute(ontology, atribute);
            }catch(Exception e){
                throw new ScriptException(e);
            }

      }
      public void sendMail(String to, String subject, String msg) throws
ScriptException{
            try{
                api.sendMail(to, subject, msg);
            }catch(Exception e){
                throw new ScriptException(e);
            }
      }
}
```

## 3.3 Variables de la Plataforma

En tiempo de ejecución el motor de Scripting disponibiliza una serie de información de forma que los scripts puedan usarlos:

| | |
|---|---|
| `ontology` | Datos de la Ontología insertada o actualizada. |
| `ontologyName` | Nombre de la Ontología insertada o actualizada. |
| `ontologyId` | Lista con lo Id de la instancia de la Ontología insertada de las instancias actualizadas. |
| `typeMessage` | Tipo de operación que ha desencadenado el Script INSERT / UPDATE. |
| `sessionKey` | SessionKey que ha realizado la inserción o la actualización. |

Estas variables están accesibles en los scripts ante eventos, no en los scripts temporizados

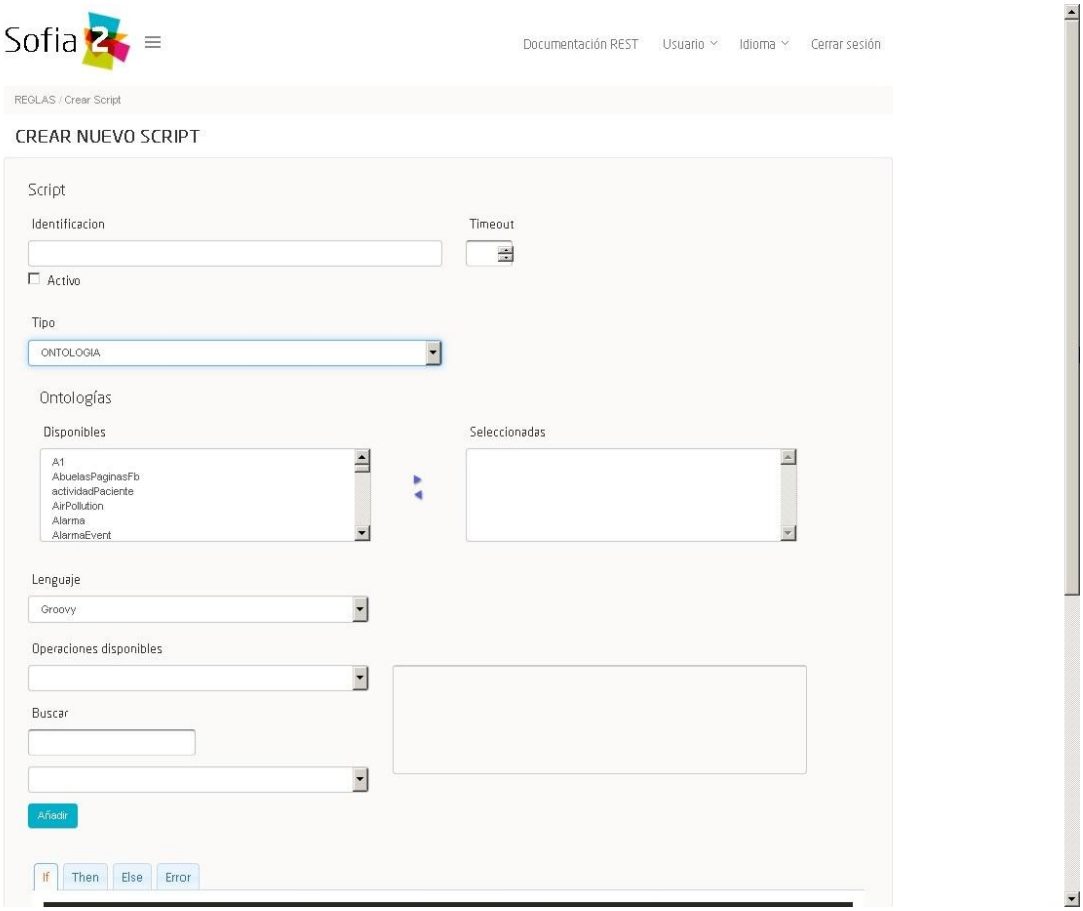
| | |
|---|---|
| `scriptName` | Nombre del Script que se ha lanzado.. |

Esta variable está disponible en todos los Scripts.

| | |
|---|---|
| `Error` | Excepción generada. |

Esta variable sólo tiene valor en los Script del bloque Error.

## 3.4 Creation of a Script




Cuando creamos un Script debemos suministrar esta información:

- **Identificación:** Valor único que identifica nuestro Script.
- **Ontología:** Ontología que provocará la ejecución del Script.
- **Activo:** para activar y desactivar el Script
- **Temporizador:** Indica cada cuanto tiempo ha de lanzarse el Script (Script Temporizado)
- **Timeout:** Tiempo máximo que se permite al script para su ejecución (si es mayor que el indicado por la plataforma se toma el de la plataforma)

En función de los parámetros de entrada un Script se convierte en un:

- **Script Temporizado:** Valor Temporizador seleccionado.
- **Script ante evento:** Valor Ontología seleccionado.

Veamos un ejemplo de un Script que se ejecuta antes la inserción o actualización de la Ontología **SensorHumedad**.


- El **bloque If**  (En Script asociados a ontología es obligatorio) define una condición que devuelva un valor true o false y que provocará la ejecución del bloque **then** o **else**.

```
api = new ApiGroovy();
texto = api.getAtribute(ontology, "SensorHumedad.identificador");
if (texto=="ST-TA114"){
    return true;
}else{
    return false;
}
```

En el ejemplo anterior, lo primero que hacemos es inicializar una de las operaciones que tenemos definidas.

```
api = new ApiGroovy();
```

Después en una variable almacenamos el valor del atributo **SensorHumedad.identificador** de la Ontología insertada.

```
texto = api.getAtribute(ontology, "SensorHumedad.identificador");
```

También podríamos haber evaluado previamente si la operación es de inserción o de actualización.

```
if (typeMessage=="INSERT"){
```

Por último y una vez tenemos los valores sobre los que queremos realizar la evaluación del bloque if, la ejecutamos.

```
if (texto=="ST-TA114"){
    return true;
}else{
    return false;
}
```

En función de la evaluación del bloque anterior, si existe (En los Script Temporizados es opcional), ejecutaremos el bloque Then o Else.

- **El bloque then:** este bloque se ejecuta cuando la condición del **If** es **true.**

```
api = new ApiGroovy();
try{
    borrado = api.rolback(ontologyName, ontologyId[0]);
    creado = api.insert(ontologyName, ontology);
    texto = api.getAtribute(ontology, "SensorHumedad.identificador");
    c = new systemprinter();
    c.print("Borrado id "+borrado);
    c.print("Creado id "+creado);
    c.print("Valor del Atributo SensorHumedad.identificador "+texto);
    c.print("Operacion Lanzada "+typeMessage);
    api.sendMail("sofia2oncloud@gmail.com", "CASO THEN", ontology);
}catch(ScriptException e){
    api.sendMail("sofia2oncloud@gmail.com", "ERROR SCRIPT", e);
}

```

  - `borrado = api.rolback(ontologyName, ontologyId[0]);` Borramos la instancia de la Ontología que se ha insertado.
  - `creado = api.insert(ontologyName, ontology);` Insertamos otra vez la Ontología que hemos insertado a través del mensaje SSAP.
  - `c = new systemprinter();` Disponibilizamos otra librería de operaciones.
  - Manejamos las excepciones que pueden ser generadas, de forma que si se produce un error enviamos un email con el asunto “ERROR SCRIPT” y la excepción que se ha generado.

```
}catch(ScriptException e){
    api.sendMail("sofia2oncloud@gmail.com", "ERROR SCRIPT", e);
}
```

- El **Bloque ELSE** es opcional y es ejecutado cuando no se cumple la condición del Bloque IF, y si se ha definido alguna operación.

```
api = new ApiGroovy();
api.sendMail("sofia2oncloud@gmail.com", "ERROR SCRIPT", error);
```

  En este caso enviamos un email indicando como asunto CASO ELSE y enviando la Ontología que ha desencadenado esta ejecución.

- El **Bloque ERROR** es opcional y se ejecuta cuando se produce un error no controlado en alguno de los casos anteriores.

  En este ejemplo envía un email indicando que se ha producido un error y el error que se ha producido.

```
api = new ApiGroovy();
api.sendMail("sofia2oncloud@gmail.com", "ERROR SCRIPT", error);
```

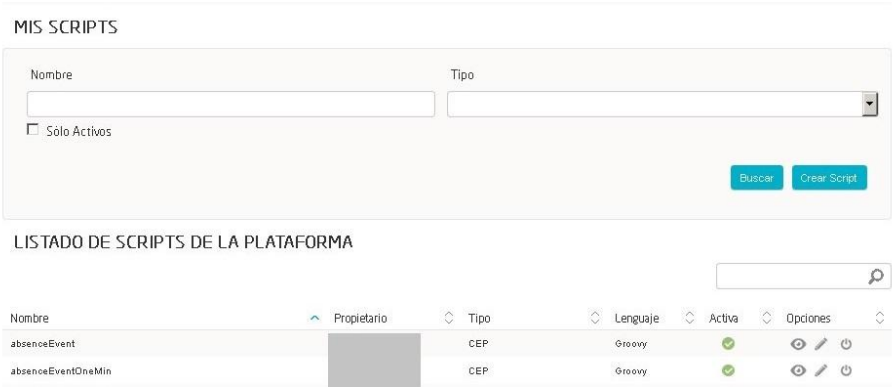
# 4 UI PARA CREACIÓN DE SCRIPTS PERSONALIZADOS

SOFIA2 permite la creación de scripts y de APIs (operaciones) desde la Consola Web de Configuración y desde su API REST.

En función del rol podremos:

- Gestionar APIs: usuario administrador
- Gestionar Scripts: usuario administrador y colaborador.

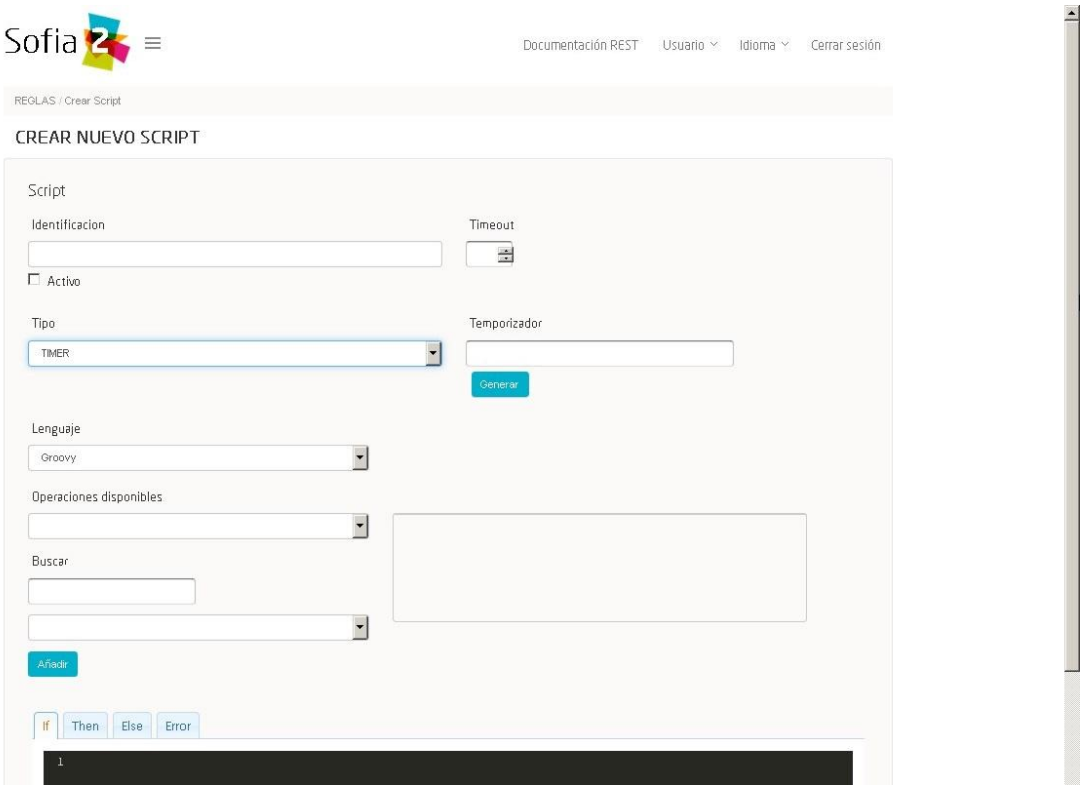
## 4.1 Listado de Scripts




Dependiendo del rol del usuario tendremos acceso a una parte:

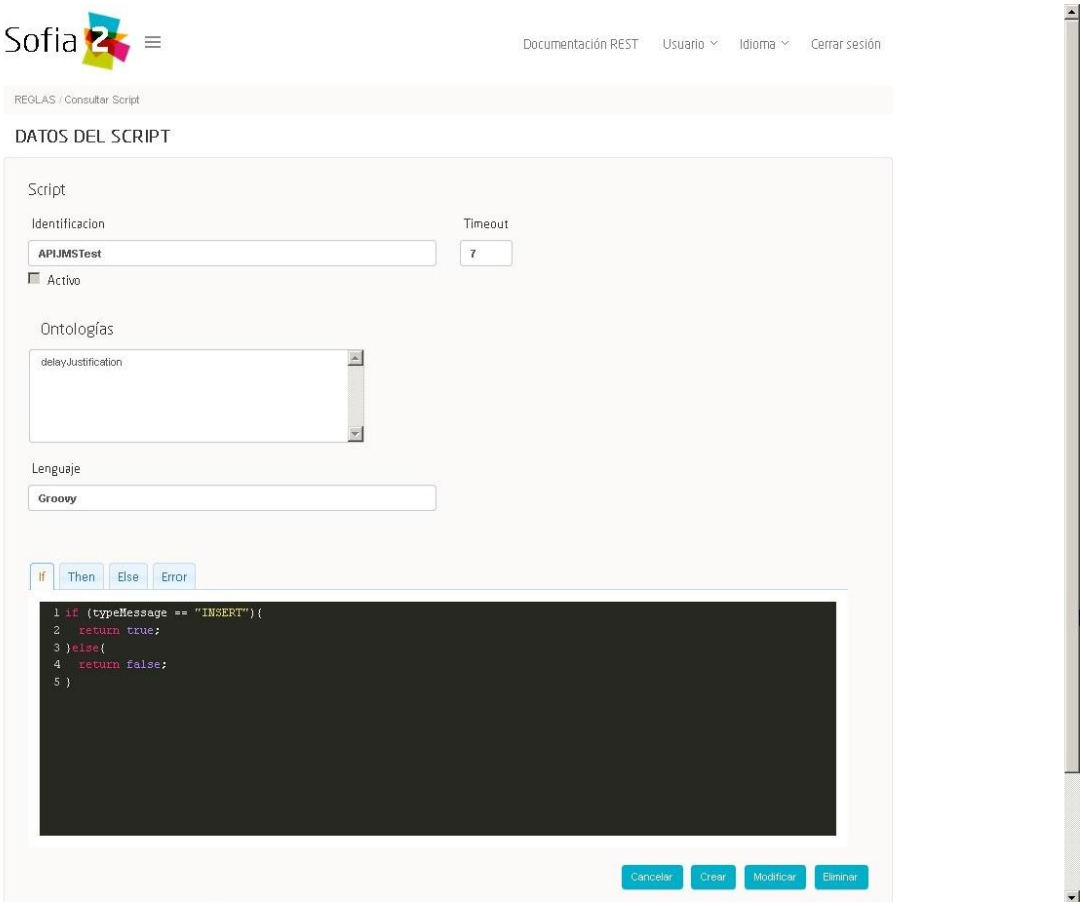
- **Los Script temporizados,** solo pueden ser creados por los Administradores
- **Los Scripts asociados a una ontología:**
  - Un administrador podrá hacer una gestión completa.
  - Un colaborador sólo podrá gestionar sus propios scripts que podrán usar ontologías sobre las que el usuario tenga permiso.
- **Los Scripts asociados a una ontología padre:**
  - Un administrador podrá hacer una gestión completa.
  - Un colaborador sólo podrá gestionar sus propios scripts que podrán usar ontologías sobre las que el usuario tenga permiso.
- **Los Scripts tipo Cep:** pueden ser creadas por usuarios con rol Administrador y Colaborador.

## 4.2 Script Temporizados



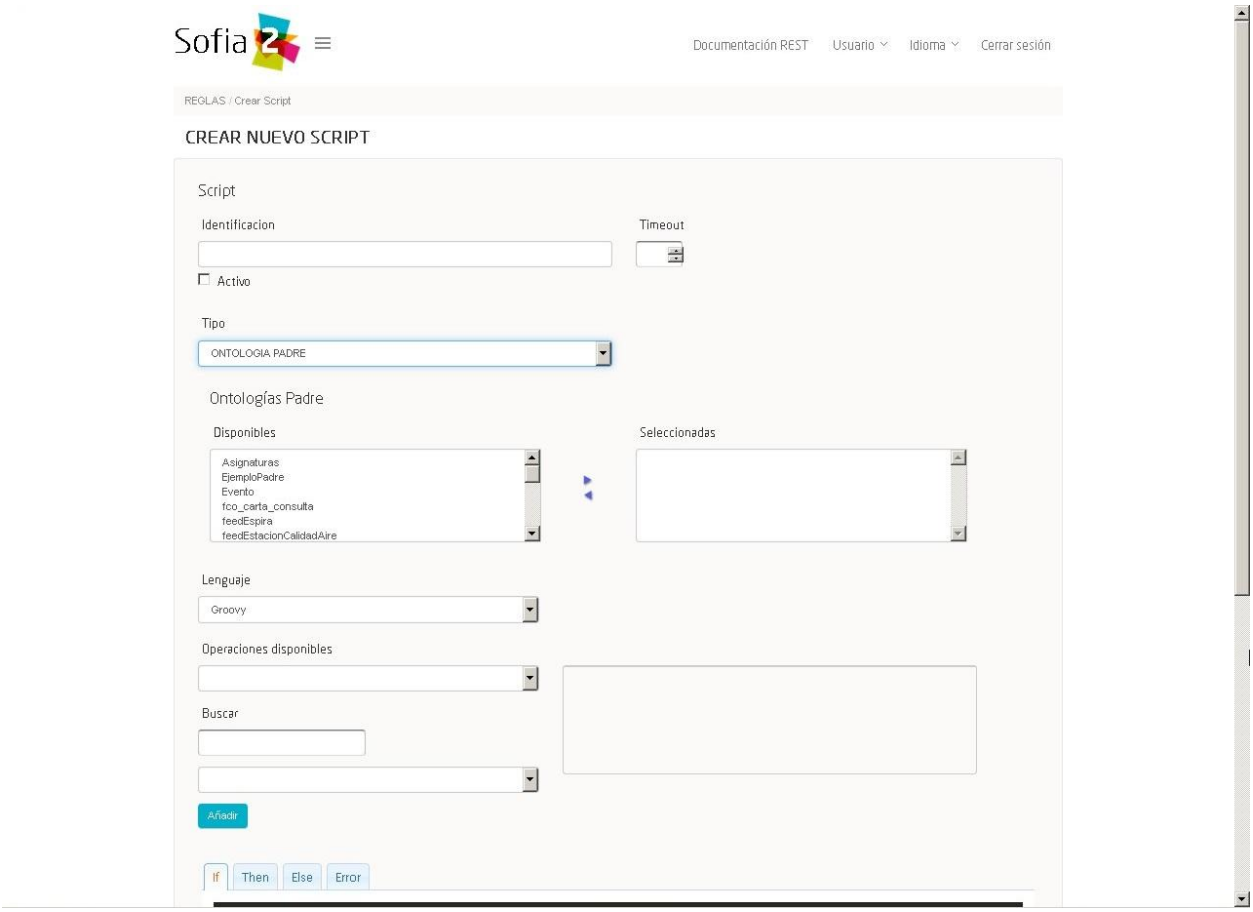

- Los Script de tipo temporizado no tienen asociada ninguna Ontología.
- Definen el tipo de temporización a través de elementos finales, finalmente esta temporización se convierte en una temporización de tipo CRON.
- Este tipo de Script han de tener definido obligatoriamente el bloque **Then**
- Opcionalmente podrán tener definido un bloque **If, Else** y **Error.**
- Si define un bloque **If** cada vez que es lanzado el Script, en función del valor indicado en el atributo Temporizador se evalúa la condición y en función de si esta se cumple se ejecuta el bloque Then o Else, si no existe condición siempre se ejecuta el bloque Then.

## 4.3 Script asociados a Ontologías



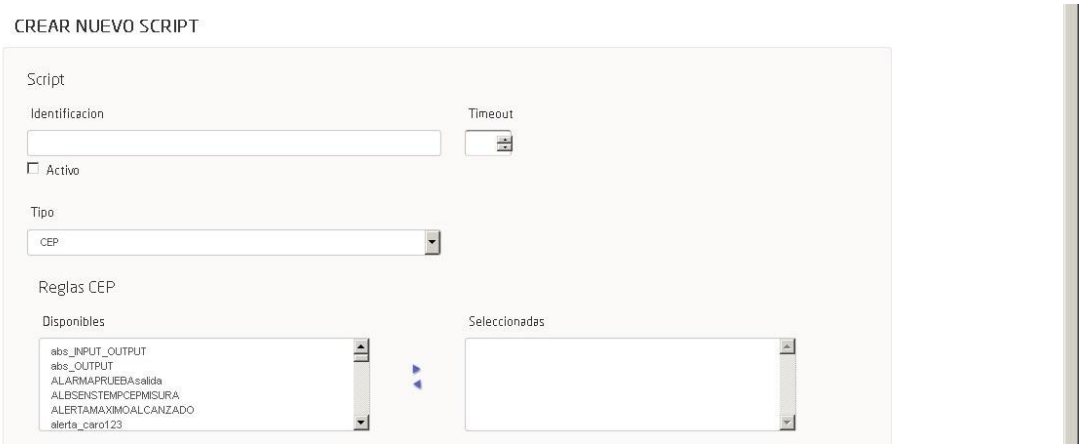

- Los Script asociados a una Ontología son lanzados cuando se realiza una inserción o una actualización de la Ontología asociada.
- Este tipo de Script han de tener definido obligatoriamente el bloque **If** y **Then**
- Opcionalmente podrán tener definido un bloque **Else** y **Error.**
- Cada vez que se realiza una inserción o una actualización de una **Ontología** se lanzan los Script asociados a estas, el primer paso es ejecutar el bloque **If,** que ha de devolver un valor true o false, si se cumple la condición se ejecuta el bloque Then, y si no se cumple y está definido el bloque Else.

## 4.4 Script asociados a Ontologías Padre




- Scripts associated to a Parent Ontology are triggered whenever there is an insert or an update on the associated Ontology.
- The blocks **If** and **Then** are mandatorily defined in this type of script.
- The blocks **Else** and **Error** can be optionally defined.
- Whenever an insert or an update of a Parent Ontology is made, the scripts associated to this Ontology are launched. The first step is executing the **If** block, which must return a true or false value. If the condition is fulfilled, the Then block is executed; if the condition is not fulfilled and the Else block is defined, then the Else block is executed.

## 4.5 Script Tipo Cep




- Los Scripts tipo Cep tienen la principal característica de que se ejecutan cada vez que se produce un evento indicado por el creador del Script.
- Este tipo de Script han de tener definido obligatoriamente el bloque **If** y **Then.**
- Opcionalmente podrán tener definido un bloque **Else** y **Error**.
- Cada vez que se ejecute el script se evaluará la condición que se indique en el campo **IF**, devolviendo el control al contenedor de Script que ha de devolver un valor true o false, si se cumple la condición se ejecuta el bloque Then, y si no se cumple y está definido el bloque Else.

# 5 SEGURIDAD EN LA EJECUCIÓN DE SCRIPTS

El módulo de Script de la plataforma SOFIA2 dispone de 2 medidas de seguridad con la finalidad de evitar que la ejecución de un Script suponga un riesgo a la integridad del sistema.

## 5.1 Seguridad Activa

Basada en Java Policy (más información sobre esta especificación en los enlaces **http://docs.oracle.com/javase/6/docs/technotes/guides/security/permissions.html** y

**http://docs.oracle.com/javase/6/docs/technotes/guides/security/PolicyFiles.html**).

Permite definir qué operaciones soportará la JVM, siendo esta la que impedirá que se ejecute código que no cumpla con las restricciones de seguridad.

Este nivel de seguridad aplica tanto a los Script como a las operaciones definidas.

La definición de un TimeOut a nivel de la plataforma asegura que ningún Script esté en ejecución más tiempo del máximo permitido, lo que provocará que el hilo en el que se lanza sea eliminado cuando pasa el tiempo máximo de ejecución.

## 5.2 Seguridad Pasiva

Basado en un analizador sintáctico, impide que cuando definimos un Script podamos hacer uso de estructuras no permitidas.

Los Scripts tienen las siguientes restricciones:

- Definir Clases.
- Definir Métodos.
- Importar Clases.
- Hacer uso de System.
- Hacer uso de las clases:

  BufferedInputStream, BufferedOutputStream, BufferedReader, BufferedWriter, ByteArrayInputStream, ByteArrayOutputStream, CharArrayReader, CharArrayWriter, Console, DataInputStream, DataOutputStream, File, FileDescriptor, FileInputStream, FileOutputStream, FilePermission, FileReader, FileWriter, FilterInputStream, FilterOutputStream, FilterReader, FilterWriter, InputStream, InputStreamReader, LineNumberInputStream, LineNumberReader, ObjectInputStream, ObjectInputStream.GetField, ObjectOutputStream, ObjectOutputStream.PutField, ObjectStreamClass, ObjectStreamField, OutputStream,

OutputStreamWriter, PipedInputStream, PipedOutputStream, PipedReader, PipedWriter, PrintStream, PrintWriter, PushbackInputStream, PushbackReader, RandomAccessFile, Reader, SequenceInputStream, SerializablePermission, StreamTokenizer, StringBufferInputStream, StringReader, StringWriter, Writer, Class, ClassLoader, Compiler, InheritableThreadLocal, Process, ProcessBuilder,ProcessBuilder.Redirect, Runtime, RuntimePermission, SecurityManager, StackTraceElement, Thread, ThreadGroup, ThreadLocal, Authenticator, CacheRequest, CacheResponse, ContentHandler, CookieHandler, CookieManager, DatagramPacket, DatagramSocket, DatagramSocketImpl, HttpCookie, HttpURLConnection, IDN, Inet4Address, Inet6Address, InetAddress, InetSocketAddress, InterfaceAddress, JarURLConnection, MulticastSocket, NetPermission, NetworkInterface, PasswordAuthentication, Proxy, ProxySelector, ResponseCache, SecureCacheResponse, ServerSocket, Socket, SocketAddress, SocketImpl, SocketPermission, StandardSocketOptions, URI, URL, URLClassLoader, URLConnection, URLDecoder, URLEncoder, URLStreamHandler.

Para hacer cualquier tipo de acción fuera de los bloques de control deben invocar las operaciones definidas que no tienen estas restricciones.

Cuando creamos un Script a través del UI se valida que no incumple estas validaciones y que puede ser almacenado en la base de datos para su ejecución.