HOW TO DEVELOP A CLIENT KP ON THE SOFIA2 PLATFORM

MAY 2016

Version 4

indra

1 INDEX

2 INTRODUCTION

2.1 Previous documentation

To understand the contents of the current document we suggest the reader to review previously the next list of documentation:

- **(EN) SOFIA2-First Steps with SOFIA2**
- **(EN) SOFIA2-How to develop on the SOFIA2 platform**
- **(EN) SOFIA2-SOFIA2 APIs**
- **(EN) SOFIA2-Web Console Use Guide**

2.2 Goal and scope of the current document

The current document describes the steps to develop a system of JavaScript KPs allow for the implementation of a notice board.

2.3 Scenario

The practical example approach described in this document is the operation of a notice board. The example is made up of two main parts:

- Firstly a JavaScript Producer KP in charge of asking the user for information on the advertisements (title and content) to then store that information.
- Secondly a JavaScript Consumer KP which will be in charge of visualizing the previously-created announcements. As the announcements are being inserted with the previous KP, this KP will show them, one after another.

3 STEPS TO DEVELOP A JAVASCRIPT PRODUCER KP ON THE PLATFORM

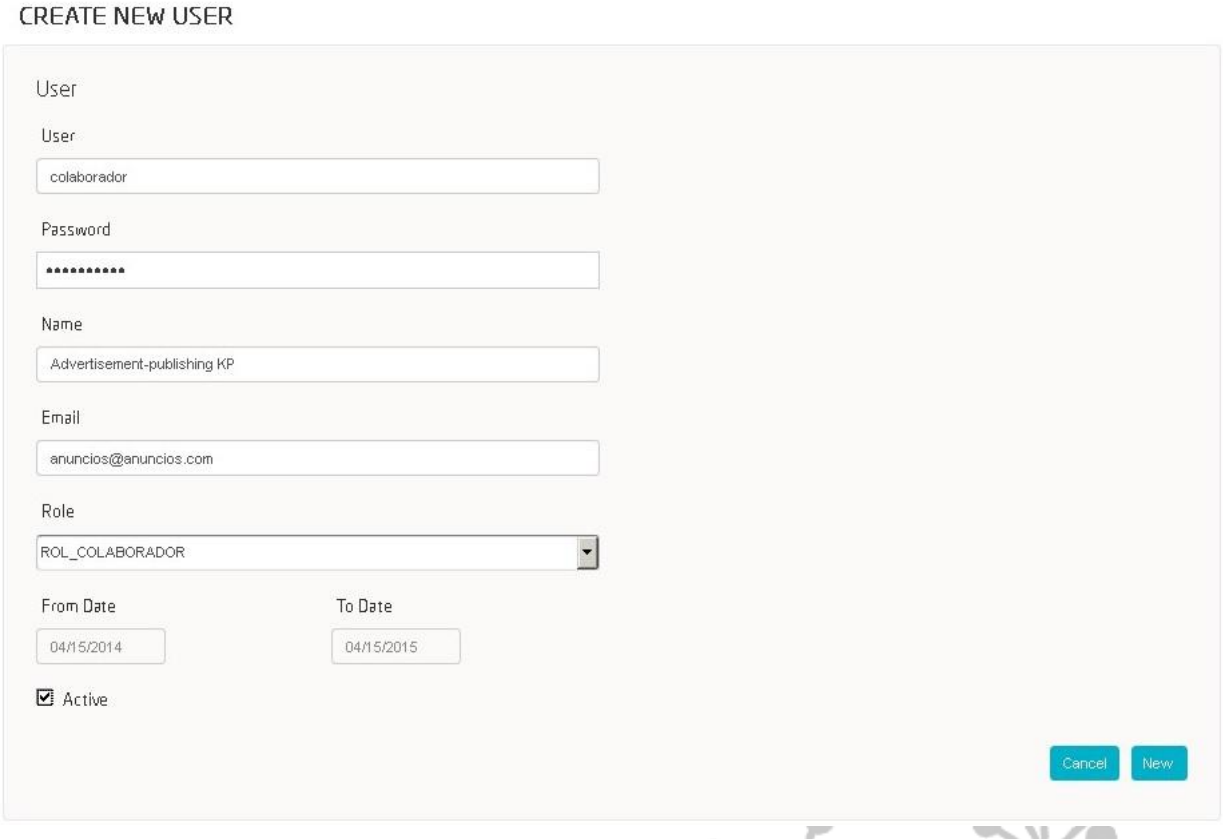
3.1 Registering in the SOFIA2 platform

We access the SOFIA2 platform using the **Web Console**. We will now detail the steps to develop the proposed example on the SOFIA2 platform.

3.1.1 Joining the platform

To develop the JavaScript KP in the example that the current document shows, we will need a user associated to the role Collaborator. Please review the guide **(EN) SOFIA2-Web Console Use Guide** to see the process to get a user with the role that the example proposes.

We register the username **"colaborador"**. Once the user is registered in the platform, the user can have access to it, according to its role.



3.2 Information ontologization

We have to register the ontology "**Anuncios**" (Spanish for Advertisements) in our specific ase. To do so, we will follow the next steps.

3.2.1 Identifying the concept's attributes

Its task is identifying the data grouped by the information concepts and which will be relevant for the KPs.

For example, for the concept **SensorAnuncio** (Advertisement sensor), we may consider the following attributes:

- **Titulo** (meaning title)**:** string
- **Timestamp:** integer
- **Contenido** (meaning contents)**:** string

3.2.2 Modelling in JSONSchema format

Once the data to be shared are identified, the next step is standardizing them so that they have a unique identification to any KP in the platform. This is the goal of the information ontologization, where each relevant concept is defined according to a JSONSchema.

For example, the concept **SensorAnuncio**, with the previously identified attributes, would be defined in JSONSchema format like this:

Corresponding to the JSONSchema of the **Anuncios** ontology.

SensorAnuncio.json

```
{
	"$schema": "http://json-schema.org/draft-04/schema#",
	"title": "Anuncios Schema",
	"type": "object",
	"required": [
		"SensorAnuncio"
	],
	"properties": {
		"_id": {
			"type": "object",
			"$ref": "#/titulo"
		},
		"SensorAnuncio": {
			"type": "string",
			"$ref": "#/datos"
		}
	},
	"additionalProperties": false,
	"titulo": {
		"title": "id",
```

```
                "description": "Titulo insertado del Anuncio",
                "type": "object",
                "properties": {
                        "$oid": {
                                "type": "string"
                        }
                },
                "additionalProperties": false
        },
        "datos": {
                "title": "datos",
                "description": "Info contenido",
                "type": "object",
                "required": [
                        "titulo",
                        "timestamp",
                        "contenido"
                ],
                "properties": {
                        "titulo": {
                                "type": "string"
                        },
                        "timestamp": {
                                "type": "object",
                                "required": [
                                        "$date"
                                ],
                                "properties": {
                                        "$date": {
                                                "type": "string",
                                                "format": "date-time"
                                        }
                                },
                                "additionalProperties": false
                        },
                        "contenido": {
                                "type": "string"
                        }
                },
                "additionalProperties": false
        }
}
```

Thus the information produced and consumed by the SensorAnuncio KPs will be normalized to the following format:

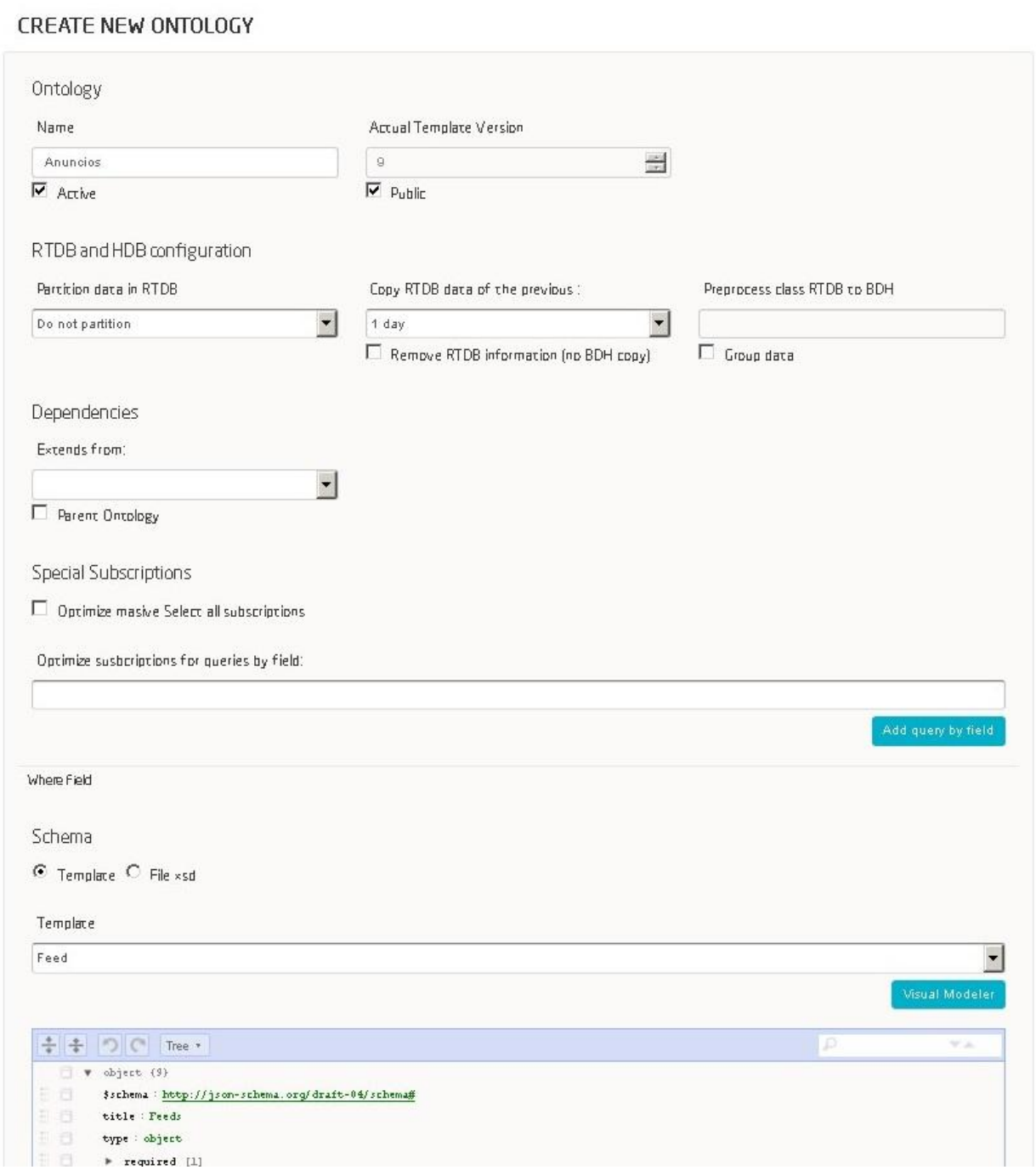
SensorAnuncio-instance.json

```
{
        "SensorAnuncio": {
                "titulo":"Coca-Cola",
                "timestamp": {
                        "$date": "2014-02-24T13:32:03.176Z"
                },
                "contenido":"Lanza su nuevo envase de botella en formato de 350ml"
        }
}
```

3.2.3 Registering the ontology in the platform

An ontology must be registered in the platform to become operational and be available for use by the KPs to insert and consume the information it describes.

In the guide **(EN) SOFIA2-Web Console Use Guide** you can see the detail on how to register ontologies. We register our ontology "**Anuncios**".



3.3 KP Development

A KP is any software that produces or consumes information to collaborate with others throughout the platform, thus forming a ***SmartSpace*** with the other KPs with which it works.

To develop a KP we must not only program its business logic, but also perform the following steps in the platform.

3.3.1 Getting authorization for proprietary user in the platform

For a user's KP to produce or consume data from a given ontology, the user must have the right permissions on that ontology.

An ontology registered in the platform may not be visible for a given user or, even if it is, the user may be limited when operating with it due to her permissions.

From the Web Console, as described in the guide **(EN) SOFIA2-Web Console Use Guide**, the administrators can access the section **Ontologies Authorizations** to manage a user's authorizations on the different registered ontologies.

The user "**colaborador**" must have all the permissions on the ontology "**Anuncios**".

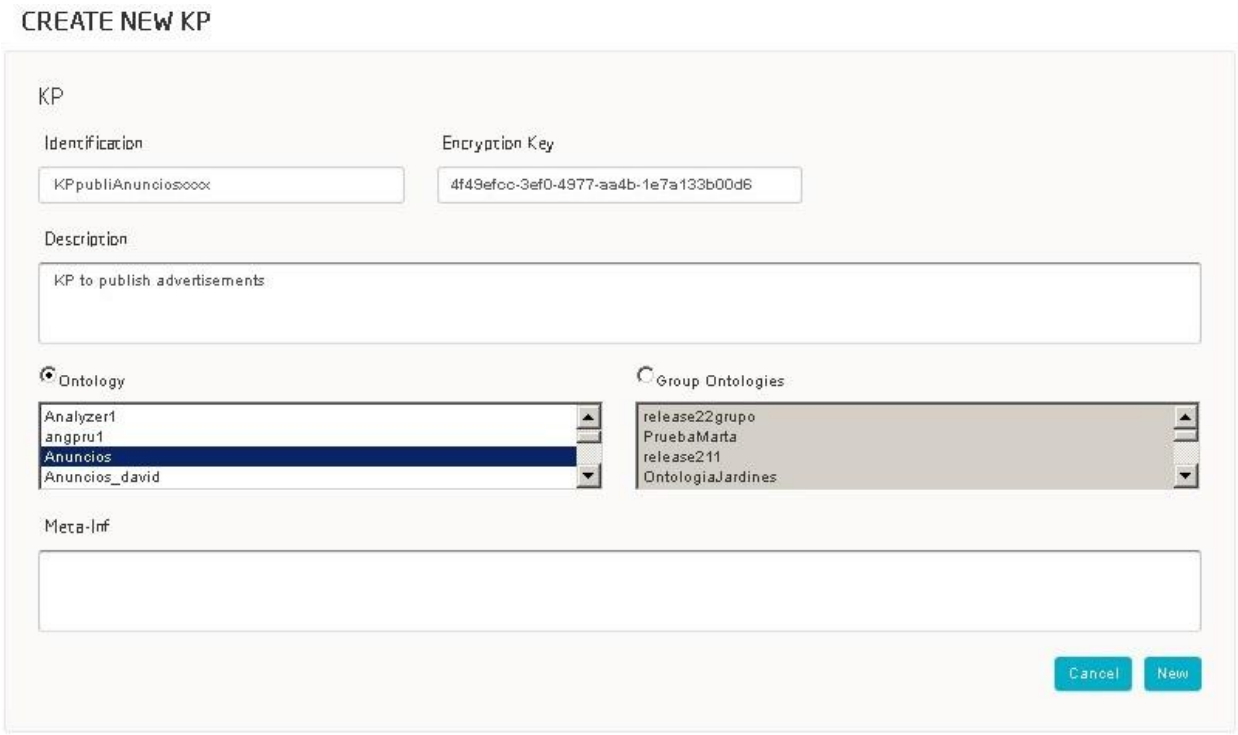
Thus, depending on the type of KP that the user will develop, the user will need permissions to **INSERT, QUERY** or **ALL** on the ontology describing the data that the KP will handle.

3.3.2 Registering KP in the platform

A user must register her KPs in the platform. Otherwise, the platform will reject the KPs' connection.

To register a KP, the platform provides an area, **KPs**, were a user can create a new KP or manage those that are already created.

We register the KP "**KPpubliAnuncios**" associated to the ontology "**Anuncios**".



As we can see, a KP can use one or several ontologies, being those the information that it will produce or consume from the platform.

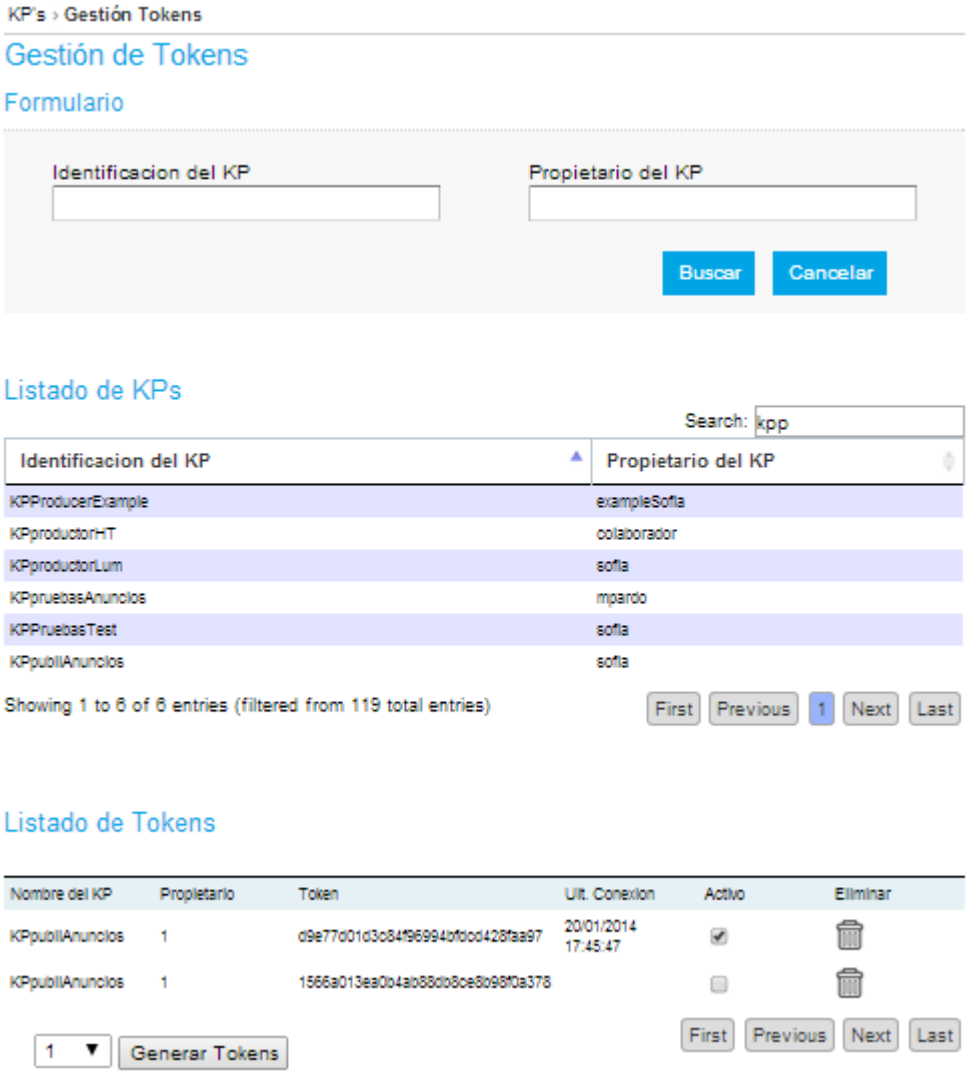
3.3.3 Managing tokens

Once the KP is registered in the platform, we need to generate a token to connect with that KP.

To do so, we will go to the section **KP's/apps SOFIA2 > My Tokens**, where the user can create one or several tokens associated to a KP (depending to the user's role's restrictions).

An administrator can create tokens on any KP registered in the platform.

A collaborator or a user can only register tokens for those KPs that they own.


KP's › Gestión Tokens
Gestión de Tokens
Formulario
Identificacion del KP
Propietario del KP
Buscar
Cancelar
Listado de KPs
Search: kpp
Identificacion del KP
Propietario del KP
KPProducerExample
exampleSofia
KPproductorHT
colaborador
KPproductorLum
sofia
KPpruebasAnuncios
mpardo
KPPruebasTest
sofia
KPpublIAnuncios
sofia
Showing 1 to 6 of 6 entries (filtered from 119 total entries)
First
Previous
1
Next
Last
Listado de Tokens
Nombre del KP
Propietario
Token
Ult. Conexion
Activo
Eliminar
KPpublIAnuncios
1
d9e77d01d3c84f96994bfbcd428faa97
20/01/2014 17:45:47
KPpublIAnuncios
1
1566a013ea0b4ab88db8ce8b98f0a378
First
Previous
Next
Last
1
Generar Tokens

4 IMPLEMENTATION OF JAVASCRIPT PRODUCER KP

We will now develop the code needed to implement a **JavaScript KP.**

In our case we will develop the KP "**KPpubliAnuncios**", that asks for the advertisement's title and content, then sends it to the SIB whenever the user presses the "Send" button.

Having created a Java - Web project, either directly or with the available infrastructure, we start developing the KP.

We have several functions available in the file "**index.jspx**":

- **conectarSIBconToken**, connect SIB with Token, which gets the KP user in contact with the platform and associates a session key.

-

```
function conectarSIBConToken(token, instance){

      joinToken(token, instance, function(mensajeSSAP){
            if(mensajeSSAP != null && mensajeSSAP.body.data !=
null && mensajeSSAP.body.ok == true){
                  $("#info").text("Conectado al sib con
sessionkey: "+mensajeSSAP.sessionKey).show();
            }else{
                  $("#info").text("Error conectando del
sib").show();
            }
      });
}
```

- **desconectarSIB**, disconnect SIB, which disconnects the KP's user from the platform.

```
function desconectarSIB() {
      leave(function(mensajeSSAP){
            if(mensajeSSAP != null && mensajeSSAP.body.data !=
null && mensajeSSAP.body.ok == true){
                  $("#info").text("Desconectado del sib").show();
            }else{
                  $("#info").text("Error desconectando del
sib").show();
            }
      });
}
```

- **enviarDatos**, send data, which sends the title and content that the user inserted when the button "Send is pressed", so that the information is stored in the RTDB.

```
function enviarDatos(titulo, contenido) {
      var timestamp = new Date().getTime();
      var queryMongo =
"{'SensorAnuncio':{'contenido':'"+contenido+"','timestamp':{'$date
':'"+timestamp.toISOString()+"'},'titulo':'"+titulo+"'}}";
      insert(queryMongo,"Anuncios");
      alert("El anuncio ha sido enviado correctamente.");
      limpiarDatos();
}
```

- **limpiarDatos**, clean data, which deletes the inserted data about title and content from the form.

```
function limpiarDatos(){
      titulo.value="";
      contenido.value="";
}
```

Some of these functions make use of other functions implemented in the file "**kp-core.js**", where the JavaScript API is implemented. Some of the later functions are:

- `function joinToken(token, instance, joinResponse);`
- `function leave(leaveResponse);`
- `function insert(data, ontology, insertResponse);`
- `function sendMessage(tipoQuery, query, cipherMessage, responseCallback);`

Besides, the file "**index.jspx**" implements the starting form to make the advertisements' inserts:

An example implementation for the JavaScript KP would be:

Index.jspx

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<div xmlns:jsp="http://java.sun.com/JSP/Page"
xmlns:spring="http://www.springframework.org/tags"
xmlns:util="urn:jsptagdir:/WEB-INF/tags/util" version="2.0">
<jsp:directive.page contentType="text/html;charset=UTF-8"/>
<jsp:output omit-xml-declaration="yes"/>
<spring:message code="label_consumanuncios_index" htmlEscape="false"
var="title"/>
  <script type='text/javascript' src='../js/kp-core.js'>;</script>
  <script type="text/javascript">

  //<![CDATA[

   var datosTH = [];
   var grafica = null, map = null;


   var literalesOK = {
```

```
        JOIN: "Connected to SIB...",
        LEAVE: "Disconnected from SIB",
        SUBSCRIBE: "Subscription initiated...",
        UNSUBSCRIBE: "Suscription stopped",
        QUERY: "Query OK"
    };
    var literalesError = {
        JOIN: "Error to connect with SIB",
        LEAVE: "Error to disconnect with SIB",
        SUBSCRIBE: "Error to start the subscription",
        UNSUBSCRIBE: "Error to stop the subscription",
        QUERY: "Query Error"
    };

    $(function (){
      dwr.engine.setActiveReverseAjax(true);
      dwr.engine.setErrorHandler(errorHandler);
      dwr.engine.setTimeout(0);
      //initGrafica();
      //cargarAPIGoogleMaps();
    });

    function errorHandler(message, ex){
      // dwr.util.setValue("error", "DWR ERROR: " + message + " - " +
dwr.util.toDescriptiveString(ex, 2), {escapeHtml:false});
      //setTimeout(function() { dwr.util.setValue("listaH", ""); }, 5000)
    }




    function conectarSIBConToken(token, instance){

        joinToken(token, instance, function(mensajeSSAP){
            if(mensajeSSAP != null && mensajeSSAP.body.data != null &&
mensajeSSAP.body.ok == true){
                $("#info").text("Conectado al sib con sessionkey:
"+mensajeSSAP.sessionKey).show();
            }else{
                $("#info").text("Error conectando del sib").show();
            }
        });
    }


    function desconectarSIB() {
        leave(function(mensajeSSAP){
            if(mensajeSSAP != null && mensajeSSAP.body.data != null &&
mensajeSSAP.body.ok == true){
                $("#info").text("Desconectado del sib").show();
            }else{
                $("#info").text("Error desconectando del sib").show();
            }
        });
    }



    function suscribirSIB(suscripcion, refresco) {
         var queryMongo = "{'SensorAnuncio.titulo': { $ne: '' }}";
```

```
        var subcriptionNotExists=subscribe(queryMongo, "Anuncios", refresco);
        if(!subcriptionNotExists){
            $("#info").text("Ya existe una suscripcion para esa
query").show();
        }
    }

    function subscriptionWellLaunchedResponse(subscriptionId,
subscriptionQuery){
        $("#info").text("Suscrito con id: "+subscriptionId+" a query:
"+subscriptionQuery).show();
    }

    function desuscribirSIB(suscripcion, valor) {
        var queryMongo = "{'SensorAnuncio.titulo': { $ne: '' }}";
        unsubscribe(queryMongo,
          function(mensajeSSAP){
            if(mensajeSSAP != null && mensajeSSAP.body.data != null &&
mensajeSSAP.body.ok == true){
                $("#info").text("Desuscrito de "+queryMongo).show();
            }else{
                $("#info").text("Error desuscribiendo del sib").show();
            }
        },
        function(error){
            if(error =="ERROR_1" || error=="ERROR_2"){
                $("#info").text("No existe suscripcion para la
query").show();
            }
        });
    }


//Recepción de notificaciones de la suscripción:
  // 1) extraer datos temp 2) query para obtener light 3) query para ontener
energia
  function indicationForSubscription(ssapMessageJson, sourceQuery) {
      alert("Se ha recibido una notificación.");
      //alert("ssapMessageJson : " + ssapMessageJson);
      var mensajeSSAP = parsearMensajeSSAP(ssapMessageJson);
      if (mensajeSSAP != null){
        try{
          // 1) Cogemos solo el primer mensaje de la notificación
          var titulo = mensajeSSAP.body.data[0].SensorAnuncio.titulo;
          var contenido = mensajeSSAP.body.data[0].SensorAnuncio.contenido;

          pintarImagen(titulo,contenido);
        }catch(err){
            //alert ("Error Notificación:" + err);
        }
      }
  }

  function pintarImagen(titulo,contenido) {
      document.getElementById("grafica").innerHTML += "TITULO : "+titulo+
"<br /> CONTENIDO : "+contenido+ "<br /> <hr />";
  }

  // Envía una query SSAP de cualquier tipo y actualiza info estado
  function enviarQuery(tipoQuery, query) {
      var mensajeSSAP = null;
```

```
      GatewayDWR.process(query, function(data) {
          GatewayDWR._path = sibServer;
          mensajeSSAP = parsearMensajeSSAP(data);
          // Ok
          if (mensajeSSAP != null && mensajeSSAP.body.data != null &&
mensajeSSAP.body.ok == true){
              switch(tipoQuery){
               case "JOIN":
                  sessionKey = mensajeSSAP.sessionKey;
                  $('#info').text(literalesOK[tipoQuery]).show();
                  break;
               case "LEAVE":
                  sessionKey = null;
                  $('#info').text(literalesOK[tipoQuery]).show();
                  break;
               case "SUBSCRIBE":
                  // La respuesta del SUBSCRIBE no va a llegar hasta que
finalice la suscripción
                  break;
               default:
                  $('#info').text(literalesOK[tipoQuery]).show();
              }
          }
          // Error
          else{
              $('#info').text(literalesError[tipoQuery]).show();
          }
      });
  }

  // Devuelve un mensaje SSAP JSON parseado a un objeto Javascript
  function parsearMensajeSSAP(mensaje) {
      try{
         return $.parseJSON(validarSSAP(mensaje));
      }catch(e){
         //alert ("Error parseo mensaje: " + e);
         return null;
      }
  }

  // Devuelve un string JSON SSAP válido
  function validarSSAP(datos) {
      return datos.replace(/\\+\"/g,"\"")
                  .replace(/(body|data)\"\s*:\s*\"({|\[)/g,"$1\":$2")

.replace(/(}|])\"\s*,\s*\"(direction|ontology|message|session|error|ok)/g,"$1
,\"$2");
  }
  //]]>

  function prueba(){

//alert("esto es una priueba de llamada la funcion");
  }

  </script>

  <style>
h4 {color:DarkCyan;}
   .dygraph-legend {
       margin-right: 4px !important;
       top: 12px !important;
```

```
        text-align: right !important;
        font-size: 1em !important;
    }

    .dygraph-ylabel {
        margin: 30px !important;
        padding: 0 !important;
        text-align: left !important;
    }

    .dygraph-y2label {
        text-align: left !important;
    }

  </style>


    <util:panel id="title" title="${title}">
      <spring:message code="application_name" htmlEscape="false" var
="app_name"/>
      <h4>Conectar / Desconectar al SIB </h4>
      <table>
           <tr style="border:none;">
              <td align="right" style="border:none;">
                  <b>Token:</b>
                  <input type="text" id="token" size="36" style="margin:8px"
value="1f84e00342914cb985373603a86fac9d"></input>
              </td>
              <td align="left" style="border:none;">
                  <b>Instance:</b>
                  <input type="text" id="instanceByToken" size="42"
style="margin:8px" value="KPconsumAnuncios:KPconsumAnuncios01"></input>
              </td>
           </tr>
           <tr style="border:none;">
              <td colspan="2" align="center" style="border:none;">
                  <input type="button" size="40" style="margin:4px"
value="Conectar SIB" onclick='conectarSIBConToken($("#token").val(),
$("#instanceByToken").val());' ></input>
                  <input type="button" size="40" style="margin:4px"
value="Desconectar SIB" onclick='desconectarSIB()'></input>
              </td>
           </tr>
      </table>

  <h4>Suscripcion</h4>
      <table>
           <tr style="border:none;">
              <td align="center" style="border:none;">
                  <b>Refresco (ms): </b>
                  <input type="text" id="refresco" size="5"
style="margin:8px" value="10000"/>
              </td>
           </tr>
           <tr style="border:none;">
              <td colspan="3" align="center" style="border:none;">
                  <input type="button" value="Suscribir" style="margin:4px"
onclick='suscribirSIB($("#suscripcion").val(),$("#refresco").val());'/>
                  <input type="button" value="Desuscribir" style="margin:4px"
onclick='desuscribirSIB($("#suscripcion").val(),$("#valor").val());'/>
              </td>
           </tr>
```

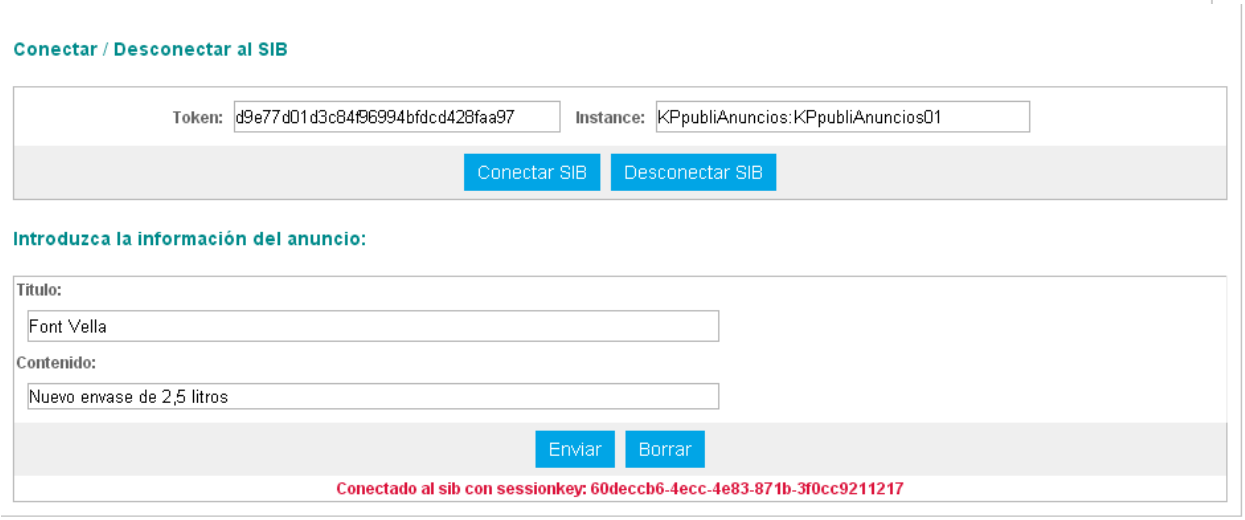
```
            <tr style="border:none;">
               <td colspan="3" align="center" style="border:none;">
               <b id="info" style="color:Crimson;margin-
top:10px;display:none;" />
               </td>
            </tr>
      </table>

    <h4>Anuncios en Tiempo Real</h4>
    <table>
            <tr>
               <td style="padding-right:2px;">
               <div id="grafica"
style="width:98%;height:auto;margin:2px;"></div></td>
            </tr>
      </table>

  </util:panel>
</div>
```

5 RUNNING THE JAVASCRIPT PRODUCER KP

1) We insert the token and the instance.
2) We click on the button “Conectar SIB” (Connect to SIB). If the connect is correctly established, we will be notified about it.
3) Then, we insert the information we want for the advertisement.



4) Finally we click on the button “Enviar” (Send).

6 STEPS TO DEVELOP A JAVASCRIPT CONSUMER KP ON THE PLATFORM

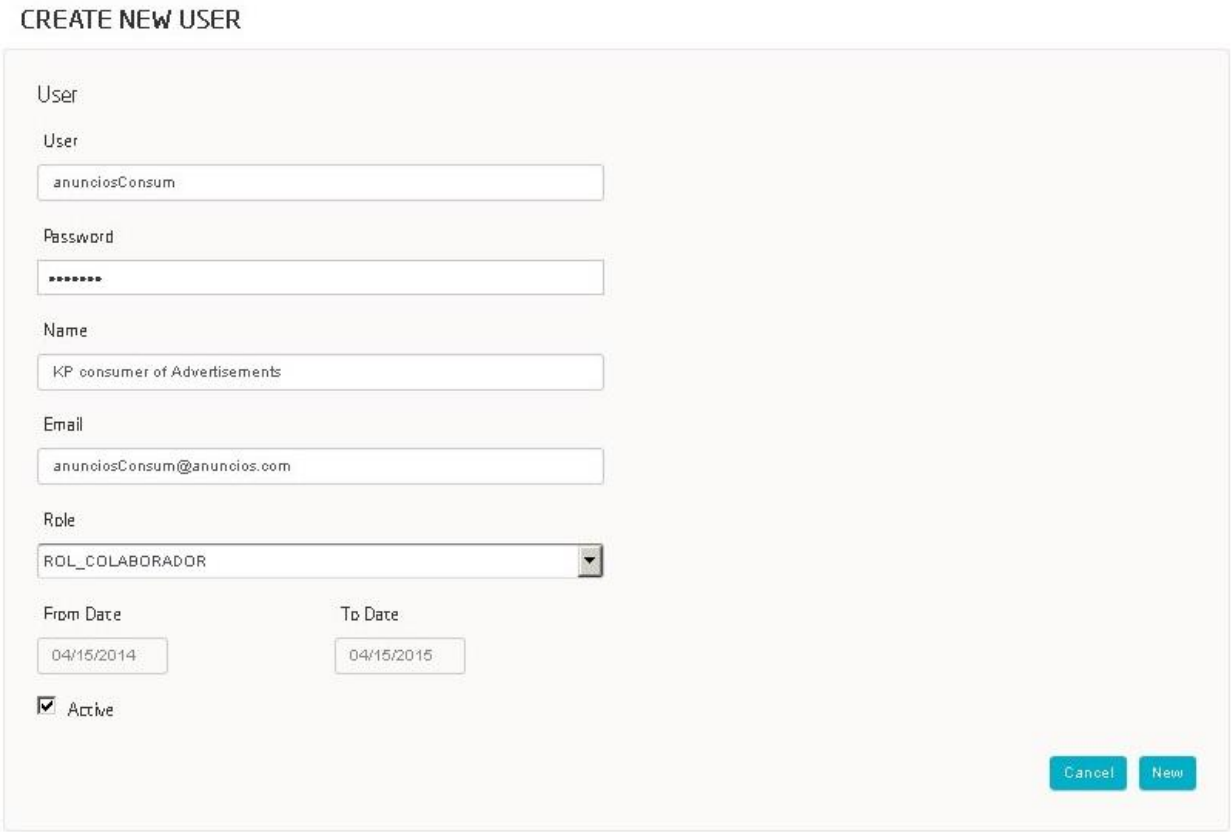
We access the SOFIA2 platform in the url **http://sofia2.com/console/login**

The steps to develop on the SOFIA2 platform are as follows:

6.1 Registering in the SOFIA2 platform

6.1.1 Joining the platform

To develop the JavaScript KP we create a new user, different from the previous one:



We register the user "**anunciosConsum**".

Once registered in the platform, the user will have access to the platform according to her role.

6.2 Information ontologization

6.2.1 Identifying the concept's attributes

This step was finished in the previous KP where we saw that the concept **SensorAnuncio** may consider the following attributes:

- **Titulo:** string
- **Timestamp:** integer
- **Contenido:** string

6.2.2 Modelling in JSONSchema format

The concept **SensorAnuncio** is already defined, with the attributes identified in the JSONSchema.

6.2.3 Registering the ontology in the platform

The ontology "**Anuncios**" is already registered, meaning that in this particular case this step is already done.

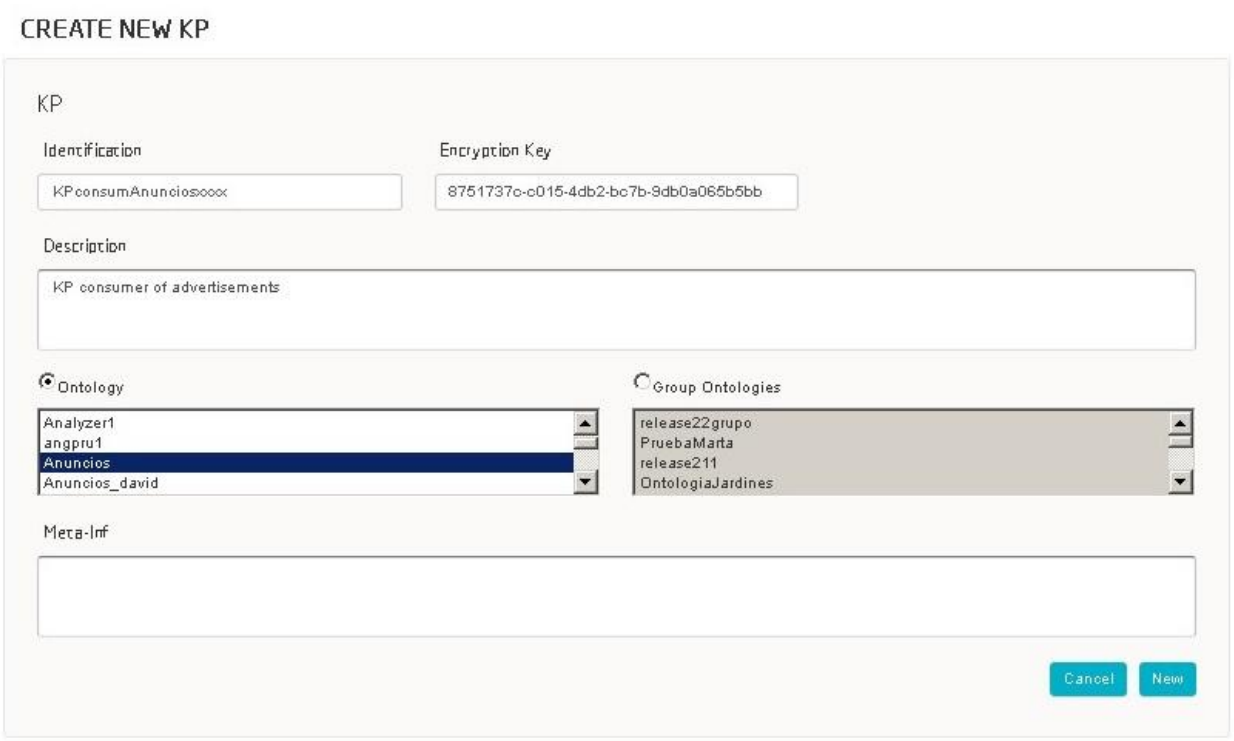
6.3 KP Development

6.3.1 Getting authorization for proprietary user in the platform

We give QUERY permissions to the user "**anunciosConsum**" on the ontology "**Anuncios**".

6.3.2 Registering KP in the platform

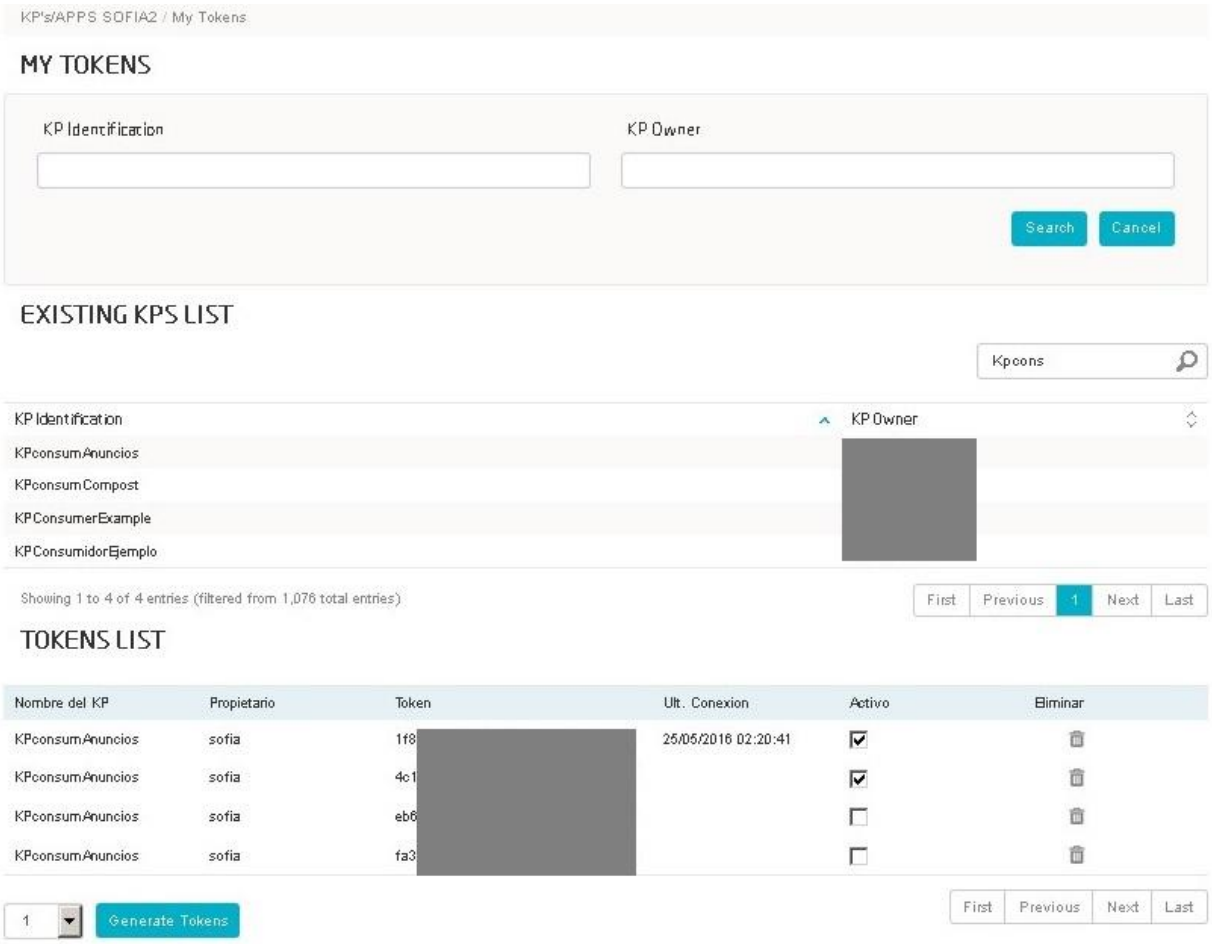
We register the KP "**KPconsumAnuncios**" associated to the ontology "**Anuncios**".



Once registered in the platform, the KP can connect to it.

6.3.3 **Managing tokens**

Once the KP is registered in the platform, we need to generate a token to connect with that KP.


KP's/APPS SOFIA2 / My Tokens
MY TOKENS
KP Identification
KP Owner
Search
Cancel
EXISTING KPS LIST
Kpcons
KP Identification
KP Owner
KPconsumAnuncios
KPconsumCompost
KPConsumerExample
KPConsumidorEjemplo
Showing 1 to 4 of 4 entries (filtered from 1,076 total entries)
First
Previous
1
Next
Last
TOKENS LIST
Nombre del KP
Propietario
Token
Ult. Conexion
Activo
Eliminar
KPconsumAnuncios
sofia
1f8
25/05/2016 02:20:41
KPconsumAnuncios
sofia
4c1
KPconsumAnuncios
sofia
eb6
KPconsumAnuncios
sofia
fa3
1
Generate Tokens
First
Previous
Next
Last

7 IMPLEMENTATION OF JAVASCRIPT CONSUMER KP

We will now develop the code needed to implement a **JavaScript KP** that consumes advertisements**.**

In our case we will develop the KP "**KPconsumAnuncios**", that receives from the SIB notifications on any announcement if the KP is subscribed to that announcement's title.

Having created a Java - Web project, either directly or with the available infrastructure, we start developing the KP.

We have several functions available in the file "**index.jspx**":

- **conectarSIBconToken**, connect SIB with Token, which gets the KP user in contact with the platform and associates a session key.

```
function conectarSIBConToken(token, instance){

      joinToken(token, instance, function(mensajeSSAP){
            if(mensajeSSAP != null && mensajeSSAP.body.data !=
null && mensajeSSAP.body.ok == true){
                  $("#info").text("Conectado al sib con
sessionkey: "+mensajeSSAP.sessionKey).show();
            }else{
                  $("#info").text("Error conectando del
sib").show();
            }
      });
}
```

- **desconectarSIB**, disconnect SIB, which disconnects the KP's user from the platform.

```
function desconectarSIB() {
      leave(function(mensajeSSAP){
            if(mensajeSSAP != null && mensajeSSAP.body.data !=
null && mensajeSSAP.body.ok == true){
                  $("#info").text("Desconectado del sib").show();
            }else{
                  $("#info").text("Error desconectando del
sib").show();
            }
      });
}
```

- **suscribirSIB**, subscribe to SIB, that subscribes to the SIB for advertisements. The subscription is made for a specific advertisement title.

```
function suscribirSIB(suscripcion, refresco) {
	var queryMongo = "{'SensorAnuncio.titulo': { $ne: '' }}";
	var subcriptionNotExists=subscribe(queryMongo,
"Anuncios",refresco);
	if(!subcriptionNotExists){
		$("#info").text("Ya existe una suscripcion para esa
	query").show();
	}
}
```

- **desuscribirSIB**, that cancels a subscription made to the SIB on a given advertisement title.

```
function desuscribirSIB(suscripcion, valor) {
	var queryMongo = "{'SensorAnuncio.titulo': { $ne: '' }}";
	unsubscribe(queryMongo, function(mensajeSSAP){
		if(mensajeSSAP != null && mensajeSSAP.body.data !=
	null && mensajeSSAP.body.ok == true){
			$("#info").text("Desuscrito de
		"+queryMongo).show();
		}else{
			$("#info").text("Error desuscribiendo del
		sib").show();
		}
	},
	function(error){
		if(error =="ERROR_1" || error=="ERROR_2"){
			$("#info").text("No existe suscripcion para la
		query").show();
		}
	});
}
```

- **subscriptionWellLaunchedResponse**, where we receive a notification about the subscription have been made correctly. This is needed due to the API.

```
function subscriptionWellLaunchedResponse(subscriptionId,
subscriptionQuery) {
	$("#info").text(
		"Suscrito con id: " + subscriptionId + " a query: "+
	subscriptionQuery).show();
}
```

- **indicationForSubscription**, where we receive the notification of the advertisements to which we have subscribed.

```
function indicationForSubscription(ssapMessageJson, sourceQuery) {
	alert("Se ha recibido una notificación.");
	var mensajeSSAP = parsearMensajeSSAP(ssapMessageJson);
	if (mensajeSSAP != null){
		try{
		// 1) Cogemos solo el 1er mensaje de la notificación
		var titulo =
	mensajeSSAP.body.data[0].SensorAnuncio.titulo;
		var contenido =
	mensajeSSAP.body.data[0].SensorAnuncio.contenido;
		pintarImagen(titulo,contenido);
		}catch(err){
			//alert ("Error Notificación:" + err);
		}
	}
}
```

- **pintarDatos**, paint data, where the web page's form's data is updated to show the advertisement's title and content.

```
function pintarDatos(titulo,contenido) {
	document.getElementById("grafica").innerHTML += "TITULO
:"+titulo+ "<br /> CONTENIDO : "+contenido+ "<br /> <hr />";

}
```

- **parsearMensajeSSAP**, parse SSAP message, returns a JSON SSAP message parsed to a JavaScript object.

```
function parsearMensajeSSAP(mensaje) {
	try {
		return $.parseJSON(validarSSAP(mensaje));
	} catch (e) {
		//alert ("Error parseo mensaje: " + e);
		return null;
	}
}
```

- **validarSSAP**, validate SSAP, returns a SSAP JSON string that can be send to the SIB.

```
function validarSSAP(datos) {
	return datos
		.replace(/\\+\"/g, "\"")
		.replace(/(body|data)\"\s*:\s*\"({|\[)/g, "$1\":$2")
		.replace(
```

```
/(}|])\"\s*,\s*\"(direction|ontology|message|session|error|ok)/g,
"$1,\"$2");
}
```

Some of these functions make use of other functions implemented in the file "**kp-core.js**", where the JavaScript API is implemented. Some of the later functions are:

- `function joinToken(token, instance, joinResponse);`
- `function leave(leaveResponse);`
- `function subscribe(suscription, ontology, refresh)`
- `function unsubscribe(querySubs, unsubscribeResponse, unsubscribeMessages)`
- `function sendMessage(tipoQuery, query, cipherMessage,responseCallback);`

Besides, the file "**index.jspx**" implements the form to make the connections, subscriptions and show the received advertisements.

An example implementation for the JavaScript KP would be:

Index.jspx

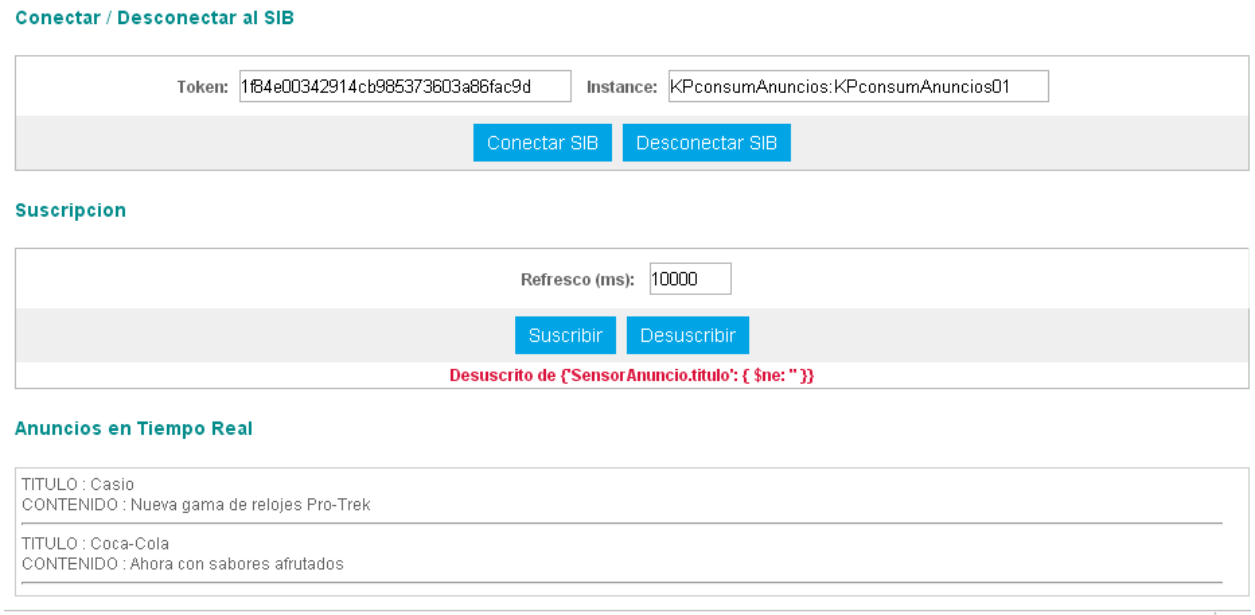
```
{
Xxx
}
```

8 RUNNING THE JAVASCRIPT CONSUMER KP

1) We insert the token and the instance.
2) We click on the button “Conectar SIB” (Connect to SIB). If the connect is correctly established, we will be notified about it.
3) We now insert the refresh time we want and click on the button Subscribe. We will be notified if the subscription has been made correctly.
4) Whenever an advertisement is inserted, the KP will receive a notification saying it.

From one of the producer KP example from the previous sections we can insert advertisements that will be visualized in this page.

5) When we want to end the subscription, we click on the button “Desuscribir”.



6) Finally we disconnect from the SIB with the button “Desconectar SIB”.