



Manual técnico del servicio Web API de transmisión automática de lecturas de contadores



Contenido

Introducción	2
Descripción de los servicios.....	3
Sección autenticación	3
DameToken	3
Sección contadores	4
EnviarLectura.....	4
CambioContador	6
Generación automática de clientes	9
Microsoft Visual Studio	9
Swagger codegen	9
Ejemplos de código consumiendo los servicios	11



Introducción

El presente documento es una guía para la utilización remota de los servicios web diseñados para la carga automática de lecturas de contadores de consumo de agua relativos a concesiones administrativas otorgadas por la Confederación Hidrográfica del Júcar.

Estos servicios deberán ser invocados por aplicaciones informáticas desarrolladas para tal fin y con el objetivo de facilitar esta labor se han elaborados siguiendo los estándares de la especificación [OpenAPI](#).

Swagger es un framework para documentar APIs Rest y el principal beneficio para un tercero que quiera consumir estas APIs es la posibilidad de automatizar la generación de los clientes que va a hacer uso de las mismas. Todo ello a través de la descripción de dichas APIs mediante ficheros JSON.

Swagger dio lugar a la especificación OpenAPI para describir APIs REST de manera uniforme e independiente del lenguaje de programación que se quiera utilizar para su consumo.

A continuación, se describirán las funcionalidades y características de esta WebAPI, así como unas directrices para su utilización desde aplicaciones que deseen hacer uso de los servicios descritos.



Descripción de los servicios

Todos los servicios que define la API se encuentran accesibles a través de la siguiente ruta:

<https://app.chj.es/ContadoresWebApi>

Para realizar pruebas puede utilizar el servicio disponible en:

<https://app.chj.es/ContadoresWebApiTest>

A continuación, se van a enumerar describiendo los parámetros de entrada, valores de salida y resultados devueltos por el servidor.

Sección autenticación

DameToken

Previo a la utilización de cualquiera del resto de servicios es necesario una autenticación.

Desde el área de dominio público del agua se les habrá previamente suministrado un identificador para cada uno de los expedientes de los que sea titular y mediante la referencia del expediente y el identificador suministrado se realiza la autenticación.

El método DameToken de la API proporciona un token JWT (JSON Web Token) que, una vez obtenido con una autenticación básica, será requerido para completar el resto de operaciones definidas en la WebAPI.

Puede consultar la documentación JWT en [JSON Web Tokens - jwt.io](https://jwt.io)

Ruta relativa: /api/Autenticacion/dametoken

Tipo de llamada: GET

Parámetros:

- Referencia (string): Referencia de un expediente de concesión de aguas
- Identificador (string): cadena previamente proporcionada de un modo privado y seguro al tenedor de derechos del expediente de referencia

Ejemplo de llamada a DameToken:

```
https://comisaria.chj.es/ContadoresWebApi/api/Autenticacion/dametoken?Referencia=2007IP9911&Identificador=91879878-d01e-4994-a31c-487675abaccc
```

Resultados desde el servidor:



```
{
  idCaptacion    integer($int32)
  idContador     integer($int32)
  fecha          string($date-time)
  valor          number($double)
  tipoIncidencia eIncidenciainteger($int32) Enum:[ 0, 1 ]
  formatoFoto    string nullable: true
  fotoBase64     string nullable: true
}
```

Ejemplo de envío sin foto:

```
{
  "idCaptacion": 6456,
  "idContador": 456,
  "fecha": "2021-07-20T10:33:38.352Z",
  "valor": 567567,
  "tipoIncidencia": 0,
  "formatoFoto": "",
  "fotoBase64": ""
}
```

Ejemplo de envío con foto:

```
{
  "idCaptacion": 6456,
  "idContador": 456,
  "fecha": "2021-07-20T10:33:38.352Z",
  "valor": 567567,
  "tipoIncidencia": 0,
  "formatoFoto": "jpg",
  "fotoBase64":
  "iVBORw0KGgoAAAANSUHEUgAAAt4AAAPTCAIAAAAIH5liAAAAAXNSR0IARs4c6QAAAAARnQU1BAACxjwv8YQ
  UAAAAJcEhZcwaADsQAAA7EAZUrDhsAAP+LSURBVHhe7J0FmCzHea7nLM70zs70MjMz89ndg3uYmUESwQaZ4
  jiGwA3dJA7ayQ3ZgRsmJw7HceIbsmM7phhlw7JY0sykvW/... (recortado por razones de
  legibilidad)"
}
```

- **idCaptacion:** Identificador de la captación correspondiente al contador (Proporcionado por CHJ)
- **idContador:** Identificador del contador (Proporcionado por CHJ)
- **fecha:** Fecha de la lectura del contador
- **valor:** Valor de la lectura
- **tipoIncidencia:** Los valores permitidos son: 0 – lectura normal. 1 – Vuelta de contador, o sea, que el contador está funcionando correctamente pero ha llegado a su máximo y ha vuelto a contar empezando por 0
- **formatoFoto:** Formato de la foto. Posibles valores “jpg”, “png”, “”. Se dejará en blanco (vacío) para indicar que no se envía foto de la lectura.
- **fotoBase64:** Foto codificada en Base64 para su envío como string. La foto no puede tener un tamaño superior a 2Mb.



Resultados desde el servidor:

- Respuesta 200 (SUCCESS):
Significa que toda ha ido correctamente y que se ha incorporado la lectura
- Respuesta 400 (BAD_REQUEST):
Significa que se ha producido un error no controlado en la petición y que no se ha incorporado la lectura
- Respuesta 401 (UNAUTHORIZED):
Significa que se ha producido un error en la autenticación mediante un JWT incorrecto
- Respuesta 403 (FORBIDDEN):
Significa que alguno de los parámetros enviados de la lecturas es incorrecto o incompatible con lecturas anteriores: volumen inferior al último registrado, fecha anterior a la última registrada, etc.
- Respuesta 404 (NOT_FOUND):
Significa que los parámetros identificando el contador producen problemas para su inserción: Referencia no válida para el JWT, Identificador de contador inexistente, Contador dado de baja, No hay captación asociada al contador, Contador no relacionado con el expediente de referencia, etc.

El cuerpo de la respuesta siempre, en caso de problemas, tendrá formato JSON con la estructura siguiente:

```
{  
  "type": string,  
  "title": string,  
  "status": integer($int32),  
  "detail": string,  
  "instance": string  
}
```

Ejemplo:

```
{  
  "type": "https://ContadoresWebApi/probs/RefIdentNotFound",  
  "title": "El contador con identificador 0 no existe",  
  "status": 404,  
  "detail": "",  
  "instance": "/api/Lecturas/enviarlectura"  
}
```

CambioContador

Este método de la API permite informar de la sustitución de un contador mediante una estructura JSON en el cuerpo de la petición.

Como respuesta de la llamada nos devolverá una estructura que contiene el identificador del nuevo contador. A partir de la llamada a “CambioContador” todos los envíos de lecturas deberán realizarse con este nuevo identificador de contador.

Ruta relativa: /api/Lecturas/cambiocontador



Tipo de llamada: POST

Cuerpo de la llamada: Encapsulado mediante JSON que responde a la siguiente estructura

```
{
  idContadorPrevio  integer($int32)
  marca              string
  modelo            string
  numeroSerie       string
  fechaInstalacion  string($date-time)
  nombre            string
  coor_utmx         number($double)
  coor_utmy         number($double)
}
```

Ejemplo:

```
{
  "idContadorPrevio": 456,
  "marca":            "SENSUS",
  "modelo":           "TYP WP DINAMIC 250",
  "numeroSerie":     "E76XM045675W",
  "fechaInstalacion": "2021-07-14T11:03:59.305Z",
  "nombre":           "El Olivar",
  "coor_utmx":       703311,
  "coor_utmy":       4363078
}
```

- **idContadorPrevio**: Identificador del contador que se va a sustituir
- **marca**: Marca del nuevo contador
- **modelo**: Modelo del nuevo contador
- **numeroSerie**: Número de serie del nuevo contador
- **fechaInstalacion**: Fecha de instalación del nuevo contador
- **nombre**: No obligatorio. Nombre del contador. Si no se especifica se le asignará al nuevo contador el del contador sustituido.
- **Coor_utmx**: No obligatorio. Coordenada X del nuevo contador. Las coordenadas tienen que estar en Datum ETRS89 Huso 30. Si el contador está en el Huso 31 se deberá re proyectar sus coordenadas para indicarlo en el Huso 30. Si no se especifica se le asignará al nuevo contador las coordenadas del contador sustituido.
- **Coor_utmy**: No obligatorio. Coordenada Y del nuevo contador. Las coordenadas tienen que estar en Datum ETRS89 Huso 30. Si el contador está en el Huso 31 se deberá re proyectar sus coordenadas para indicarlo en el Huso 30. Si no se especifica se le asignará al nuevo contador las coordenadas del contador sustituido.

Resultados desde el servidor:

- **Respuesta 200 (SUCCESS)**:
Significa que toda ha ido correctamente y que se ha procedido al alta del nuevo contador. En el cuerpo de la respuesta se encapsula un JSON con el resultado del nuevo identificador del contador dado de alta y el siguiente formato:



```
{  
  idNuevoContador integer($int32)  
}
```

- Respuesta 400 (BAD_REQUEST):
Significa que se ha producido un error no controlado en la petición y que no se ha podido proceder al cambio de contador
- Respuesta 401 (UNAUTHORIZED):
Significa que se ha producido un error en la autenticación mediante un JWT incorrecto
- Respuesta 403 (FORBIDDEN):
Significa que alguno de los parámetros enviados para la sustitución es incorrecto o incompatible con la información existente en el sistema: Fecha de instalación incorrecta, Algún parámetro ausente, etc.
- Respuesta 404 (NOT_FOUND):
Significa que los parámetros identificando el contador previo producen problemas para su sustitución: Referencia no válida para el JWT, Identificador de contador inexistente, Contador dado de baja, No hay captación asociada al contador, Contador no relacionado con el expediente de referencia, etc.

El cuerpo de la respuesta siempre, en caso de problemas, tendrá formato JSON con la estructura siguiente, ya mencionada en el apartado de envío de lecturas:

```
{  
  "type":    string,  
  "title":   string,  
  "status":  integer($int32),  
  "detail":  string,  
  "instance": string  
}
```



Generación automática de clientes

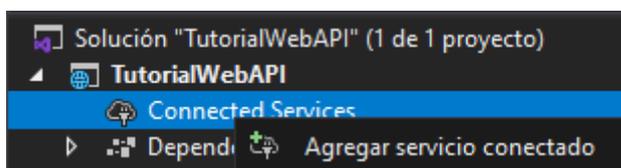
El estándar OpenAPI facilita la creación de clientes mediante un fichero JSON que define la funcionalidad de los servicios. Este fichero JSON puede ser descargado desde la siguiente URL

<https://app.chj.es/ContadoresWebApi/swagger/v1/swagger.json>

Microsoft Visual Studio

Se va a describir el proceso de generación de este cliente para una solución web de MS Visual Studio 2019. El procedimiento deber ser similar para otras plataformas o lenguajes de programación.

Lo primero es agregar un servicio conectado desde el explorador de soluciones de VS



Una vez seleccionada la opción de agregar servicio, procederemos a seleccionar como referencia de servicios OpenAPI el fichero swagger.json descargado desde la URL y Visual Studio generará los fuentes necesarios del cliente que consumirá los servicios.

Concretamente, se habrá generado en la siguiente ruta del proyecto:

```
obj\swaggerClient.cs
```

Todas las clases, atributos y métodos presentes en el cliente estarán accesibles para la aplicación.

Swagger codegen

Como ya se ha mencionado en la sección anterior, existe la posibilidad de generar automáticamente clientes para gran cantidad de lenguajes de programación. Para ello han desarrollado la herramienta java Swagger codegen que permitirá generar el código fuente de los clientes en el lenguaje que se elija.

A continuación, se describe un ejemplo de generación de cliente con esta herramienta:

Se requiere descargar la aplicación java empaquetada jar desde la URL

<https://mvnrepository.com/artifact/io.swagger.codegen.v3/swagger-codegen-cli/3.0.27>



Se puede crear un fichero config.json de configuración definiendo los espacios de nombre de la aplicación que va a utilizar el cliente.

```
{  
  "modelPackage": "WAPIContadores",  
  "apiPackage": "WAPIContadores"  
}
```

Finalmente ejecutando el comando

```
java -jar .\swagger-codegen-cli.jar generate -i .\swagger.json -l csharp -o c# -c .\config.json
```

En la carpeta c# (previamente creada) se generará una solución que dará lugar a una dll IO.Swagger.dll con todas sus dependencias tras la compilación.

El parámetro -l indica el lenguaje para el que se quiere crear el cliente, en este caso c#.



Ejemplos de código consumiendo los servicios

Consumir los servicios una vez generada la clase cliente es simple. La única complejidad reside en proporcionar el token JWT a las llamadas a la API.

El asistente de Visual Studio crea las clases necesarias para invocar los servicios web definidos en el archivo swagger.json, pero no tiene en cuenta que el servicio está securizado mediante JWT.

Por tanto, tendremos que añadir el código necesario para poder pasar el token JWT, para ello crearemos una clase parcial de la misma clase generada por el asistente de VS.

En esta clase parcial crearemos un método con la firma

```
PrepareRequest(System.Net.Http.HttpClient client, System.Net.Http.HttpRequestMessage request, string url)
```

Este método lo llama el cliente de swagger cada vez que tiene que hacer una llamada al web api. En el método lo que haremos es añadir el Header "Authorization" al request con el contenido del token anteponiéndole el término "Bearer ", (importante el espacio en blanco entre Bearer y el token)

El contenido de la parcial class podría ser el siguiente:

```
namespace TutorialWebAPI
{
    public partial class swaggerClient
    {
        public ResultadoDameToken token { get; set; }

        partial void PrepareRequest(System.Net.Http.HttpClient client,
            System.Net.Http.HttpRequestMessage request, string url)
        {
            if (token != null && client.DefaultRequestHeaders.Authorization == null)
            {
                client.DefaultRequestHeaders.Add("Authorization", "Bearer " + token.Jwt);
            }
        }
    }
}
```

Y este podría ser un ejemplo de cómo subir una lectura de un contador desde la aplicación

```
ResultadoDameToken Token = null;
swaggerClient cliente;
ProblemDetails dameTokenError = null;
ProblemDetails enviarLecturaError = null;
ProblemDetails cambioContadorError = null;

dameTokenError = null;

try
{
    var http = new System.Net.Http.HttpClient();
    cliente = new swaggerClient(http);

    cliente.BaseUrl = "https://comisaria.chj.es/ContadoresWebApi";
    Token = await cliente.DametokenAsync("XXXXXXXXXX", "XXXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXXX");
    cliente.token = Token;

    DatosLectura dl = new DatosLectura();
```



```
d1.Fecha = System.DateTimeOffset.Now;
d1.IdContador = 37;
d1.IdCaptacion=4564;
d1.Valor = 5000000;
d1.formatoFoto = "";
d1.fotoBase64 = "";
await cliente.EnvialecturaAsync(d1);
}
catch (ApiException<ProblemDetails> api)
{
    await context.Response.WriteAsync(api.Result.Title + "\n" + api.Result.Detail);
}
catch (Exception ex)
{
    await context.Response.WriteAsync(ex.Message);
}
await context.Response.WriteAsync("Success!");
```