Artículo
Ricardo Paiva · 12 nov, 2021 Lectura de 21 min

Open Exchange

Curso de formación sobre Ensemble / Interoperabilidad

Este curso de formación está dirigido a todas las personas interesadas en conocer el *framework* de Interoperabilidad de IRIS. Utilizaremos Docker y VSCode.

GitHub: https://github.com/grongierisc/formation-template

1. Formación en Ensemble/Interoperabilidad

El objetivo de esta formación es aprender el *framework* de interoperabilidad de InterSystems, y en particular el uso de:
* Producciones
* Mensajes
* *Business Operations*
* Adaptadores
* *Business Processes*
* *Business Services*
* Operaciones y servicios REST

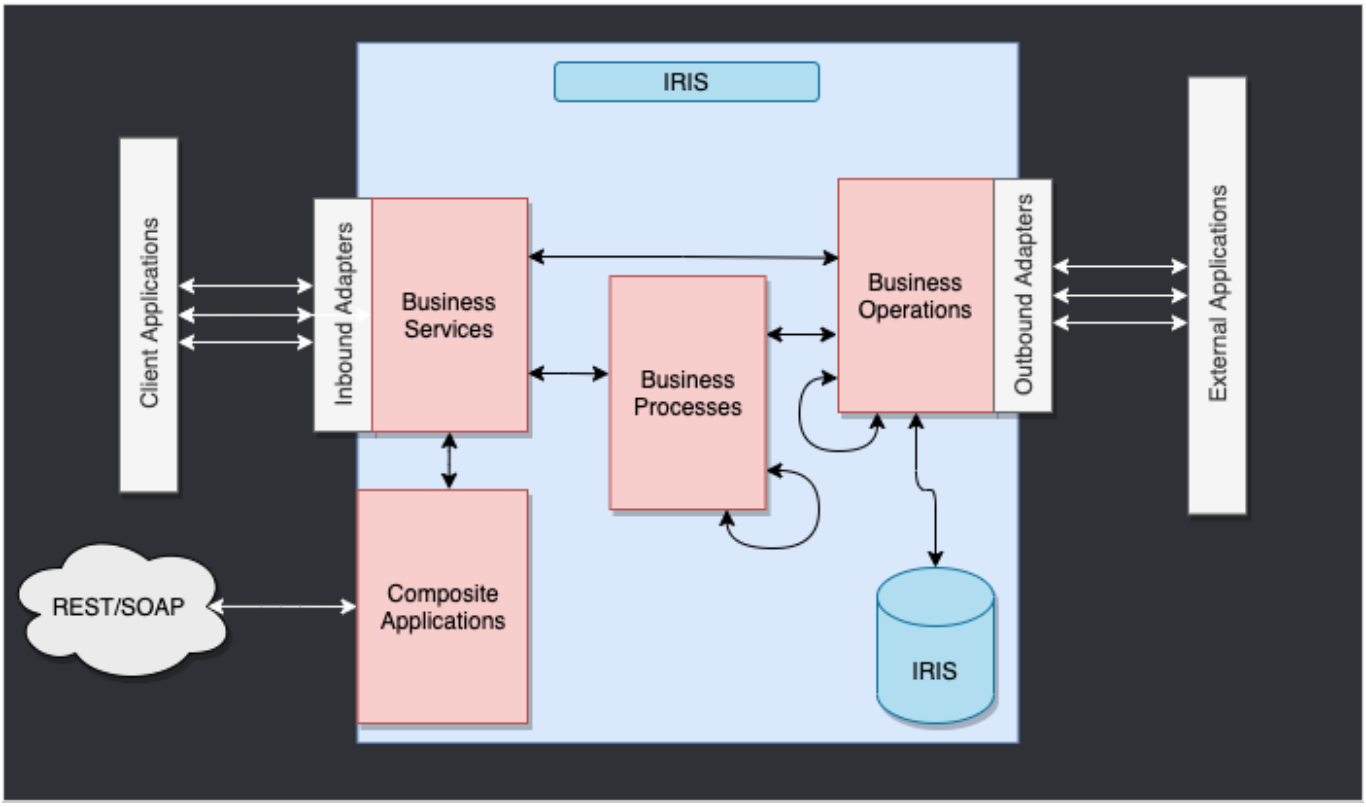
ÍNDICE:

- 1. **Formación de Ensemble/Interoperabilidad**
- 2. *Framework*
- 3. Adaptación del *framework*
- 4. Requisitos previos
- 5. Configuración
  - 5.1. Contenedores de Docker
  - 5.2. Portal de Administración
  - 5.3. Guardar el progreso
- 6. Producciones
- 7. Operaciones
  - 7.1. Creación de nuestra clase de almacenamiento
  - 7.2. Creación de nuestra clase para mensajes
  - 7.3. Creación de nuestra operación
  - 7.4. Cómo añadir la operación a la producción
  - 7.5. Pruebas
- 8. *Business processes*
  - 8.1. *Business processes* simples
  - 8.1.1. Creación del proceso
  - 8.1.2. Modificación del contexto de un *business process*
  - 8.2. Cómo hacer que el *business process* lea líneas CSV
  - 8.2.1. Creación de un mapa de registro
  - 8.2.2. Creación de una transformación de datos
  - 8.2.3. Añadir la transformación de datos al *business process*
  - 8.2.4. Configuración de la producción
  - 8.2.5. Pruebas
- 9. Obtener acceso a una base de datos externa usando JDBC
  - 9.1. Creación de nuestra nueva operación

- 9.2. Configuración de la producción
- 9.3. Pruebas
- 9.4. Ejercicio
- 9.5. Solución
- 10. Servicio REST
  - 10.1. Creación del servicio
  - 10.2. Añadir nuestro *business service* (BS)
  - 10.3. Pruebas
- Conclusión

2. *Framework*

Este es el *framework* de IRIS.



Los componentes que están en el interior de IRIS representan una producción. Los adaptadores de entrada y de salida nos permiten utilizar diferentes tipos de formato como entrada y salida para nuestra base de datos. Las aplicaciones compuestas nos darán acceso a la producción a través de aplicaciones externas como los servicios REST.

Las flechas que están entre todos estos componentes son **mensajes**. Estos pueden ser solicitudes o respuestas.

3. Adaptación del *framework*

En nuestro caso, leeremos las líneas desde un archivo CSV y las guardaremos en la base de datos IRIS.

Entonces, añadiremos una operación que nos permitirá guardar los objetos en una base de datos externa, utilizando JDBC. Esta base de datos se ubicará en un contenedor de Docker, utilizando postgre.

Por último, veremos cómo utilizar aplicaciones compuestas para insertar nuevos objetos en nuestra base de datos, o para consultar esta base de datos (en nuestro caso, a través de un servicio REST).

El *framework* adaptado a nuestro propósito nos ofrece:

4. Requisitos previos

Para esta formación, necesitarás:
* VSCode: https://code.visualstudio.com/
* El conjunto de *addons* de InterSystems para VSCode: https://intersystems-community.github.io/vscode-objectscript/installation/
* Docker: https://docs.docker.com/get-docker/
* El *addon* de Docker para VSCode

5. Configuración

5.1. Contenedores de Docker

Para tener acceso a las imágenes de InterSystems, hay que ir a esta URL: http://container.intersystems.com. Después de iniciar sesión con nuestras credenciales de InterSystems, obtendremos nuestra contraseña para conectarnos al registro. En el *addon* de Docker para VSCode, que se encuentra en la pestaña de imágenes, hacemos clic en Conectar Registro, introducimos la misma URL que antes (http://container.intersystems.com) como registro genérico y se nos pedirá que demos nuestras credenciales. El inicio de sesión es el habitual pero la contraseña es la que obtuvimos del sitio web.

A partir de ahí, deberíamos ser capaces de crear y componer nuestros contenedores (con los archivos docker-compose.yml y Dockerfile que se nos dieron).

5.2. Portal de Administración

Abriremos un Portal de Administración. Esto nos dará acceso a una página web desde la que podremos crear nuestra producción. El portal debe estar ubicado en la URL: http://localhost:52775/csp/sys/UtilHome.csp ?$NAMESPACE=IRISAPP. Necesitarás las siguientes credenciales:

LOGIN: SuperUser

PASSWORD: SYS

5.3. Guardar el progreso

Una parte de las cosas que haremos se guardarán localmente, pero todos los procesos y producciones se guardan en el contenedor de Docker. Con el fin de conservar todo nuestro progreso, necesitamos exportar todas las clases que se crean desde el Portal de Administración con ayuda del *addon* ObjectScript de InterSystems:

Tendremos que guardar de esta forma nuestra producción, mapa de registros, *business processes* y transformaciones de datos. Después de hacerlo, cuando cerremos nuestro contenedor Docker y hagamos la compilación nuevamente, aún tendremos todo nuestro progreso guardado de forma local (por supuesto, hay que hacer esto después de cada cambio que hagamos a través del portal). Para que sea accesible a IRIS de nuevo, tenemos que compilar los archivos exportados (cuando los guardemos, los *addons* de InterSystems se encargarán del resto).

6. Producciones

Ahora podemos crear nuestra primera producción. Para hacerlo, nos moveremos por los menús [Interoperability] y [Configure]:

Ahora hacemos clic en [New], seleccionamos el paquete [Formation] y elegimos un nombre para nuestra producción:

Inmediatamente después de crear nuestra producción, hay que hacer clic en la opción [Production Settings], situada encima de la sección [Operations]. En el menú de la barra lateral derecha, tendremos que activar la opción [Testing Enabled] en la sección [Development and Debugging] de la pestaña [Settings] (no te olvides de hacer clic en [Apply]).

En esta primera producción añadiremos ahora las *business operations*.

7. Operaciones

Una *business operation* (BO) es un tipo de operación específica que nos permitirá enviar solicitudes desde IRIS hacia una aplicación/sistema externo. También se puede utilizar para guardar lo que queramos directamente en IRIS.

Crearemos esas operaciones de forma local, es decir, en el archivo Formation/BO/. Cuando guardemos los archivos los compilaremos en IRIS.

En nuestra primera operación, guardaremos el contenido de un mensaje en la base de datos local.

Para hacerlo, primero necesitamos tener una forma de almacenar este mensaje.

7.1. Creación de nuestra clase de almacenamiento

En IRIS, las clases de almacenamiento extienden el tipo %Persistent. Se guardarán en la base de datos interna.

En nuestro archivo Formation/Table/Formation.cls tenemos lo siguiente:

```
Class Formation.Table.Formation Extends %Persistent
{

Property Name As %String;

Property Salle As %String;

}
```

Ten en cuenta que al guardar, de forma automática se añaden líneas adicionales al archivo. Son obligatorias y las añaden los *addons* de InterSystems.

7.2. Creación de nuestra clase para mensajes

Este mensaje contendrá un objeto Formation, situado en el archivo Formation/Obj/Formation.cls:

```
Class Formation.Obj.Formation Extends (%SerialObject, %XML.Adaptor)
{

Property Nom As %String;

Property Salle As %String;
```

```
}
```

La clase Message utilizará el objeto Formation, src/Formation/Msg/FormationInsertRequest.cls:

```
Class Formation.Msg.FormationInsertRequest Extends Ens.Request
{

Property Formation As Formation.Obj.Formation;

}
```

7.3. Creación de nuestra operación

Ahora que ya tenemos todos los elementos que necesitamos, podemos crear nuestra operación, en el archivo Formation/BO/LocalBDD.cls:

```
Class Formation.BO.LocalBDD Extends Ens.BusinessOperation
{

Parameter INVOCATION = "Queue";

Method InsertLocalBDD(pRequest As Formation.Msg.FormationInsertRequest, Output pRespo
nse As Ens.StringResponse) As %Status
{
    set tStatus = $$$OK

    try{
        set pResponse = ##class(Ens.Response).%New()
        set tFormation = ##class(Formation.Table.Formation).%New()
        set tFormation.Name = pRequest.Formation.Nom
        set tFormation.Salle = pRequest.Formation.Salle
        $$$ThrowOnError(tFormation.%Save())
    }
    catch exp
    {
        Set tStatus = exp.AsStatus()
    }

    Quit tStatus
}

XData MessageMap
{
<MapItems>
    <MapItem MessageType="Formation.Msg.FormationInsertRequest">
        <Method>InsertLocalBDD</Method>
    </MapItem>
</MapItems>
}

}
```

El MessageMap nos proporciona el método que debemos lanzar, dependiendo del tipo de solicitud (el mensaje que se envió a la operación).

Como podemos ver, si la operación recibió un mensaje del tipo Formation.Msg.FormationInsertRequest, se llamará al método InsertLocalBDD. Este método guardará el mensaje en la base de datos local de IRIS.

7.4. Cómo añadir la operación a la producción

Ahora necesitamos añadir esta operación a la producción. Para hacerlo, utilizaremos el Portal de Administración. Al hacer clic en el signo [+] junto a [Operations], tendremos acceso al [Business Operation Wizard]. Allí, elegiremos la clase de la operación que acabamos de crear en el menú desplegable.

7.5. Pruebas

Si hacemos doble clic en la operación podremos activarla. Después de hacerlo, al seleccionar la operación e ir a las pestañas [Actions] que están en el menú de la barra lateral derecha, deberíamos poder probar la operación (si no ves la sección para crear la producción, puede que tengas que iniciar la producción si se encuentra detenida).

De este modo, enviaremos a la operación un mensaje del tipo que declaramos anteriormente. Si todo sale bien, los resultados deberían ser similares a los que se muestran a continuación:

Mostrar el registro visual nos permitirá ver lo que ocurrió entre los procesos, servicios y operaciones. Aquí, podemos ver el mensaje que se envía a la operación por parte del proceso, y a la operación cuando envía de vuelta una respuesta (que en este caso solo es una cadena vacía).

8. *Business Processes*

Los *business processes* (BP) son la lógica empresarial de nuestra producción. Se utilizan para procesar las solicitudes o retransmitirlas a otros componentes de la producción.

Los *business processes* se crean dentro del Portal de Administración:

8.1. *Business processes* simples

8.1.1. Creación del proceso

Ahora estamos en el Diseñador de *Business Process*. Vamos a crear un *business process* simple que llamará nuestra operación:

8.1.2. Modificación del contexto de un *business process*

Todos los *business processes* tiene un **contexto**. Se compone de una clase para la solicitud, la clase de la entrada, una clase para la respuesta y la clase de la salida. **Los *business processes* solo tienen una entrada y una salida**. También es posible agregar propiedades.

Como nuestro *business process* solo se utilizará para llamar a nuestra *business operation*, podemos poner la clase del mensaje que hemos creado como clase para la solicitud (no necesitamos una salida ya que solo queremos insertarlo en la base de datos).

Ahora, elegiremos el objetivo de la función Call: nuestra *business operation*. Esa operación, al ser **llamada** tiene una propiedad **callrequest**. Necesitamos vincular esa propiedad callrequest con la solicitud del *business process* (ambos pertenecen a la clase Formation.Msg.FormationInsertRequest) y para ello, hacemos clic en la función Call y utilizaremos el creador de peticiones:

Ahora podemos guardar este *business process* (en el paquete 'Formation.BP' y, por ejemplo, con el nombre 'InsertLocalBDD' o 'Main'). Al igual que las operaciones, pueden crearse instancias de los procesos y se pueden probar mediante la Configuración de la producción, aunque para esto necesitan compilarse previamente (en la pantalla del *Business Process Designer*).

Por ahora, nuestro proceso solo pasa el mensaje a nuestra operación. Vamos a aumentar la complejidad para que el business process tome como entrada una línea de un archivo CSV.

8.2. Cómo hacer que el *business process* lea líneas CSV

8.2.1. Creación de un mapa de registro

Para leer un archivo y poner su contenido en otro archivo, necesitamos un Mapa de Registros (RM). En el menú [Interoperability > Build] del Portal de Administración hay un servicio para mapear los registros, especializado en archivos CSV:

Crearemos el servicio para elaborar mapas de esta manera:

Ahora deberías tener el siguiente Mapa de Registros:

Ahora que el mapa está creado, debemos generarlo (con el botón *Generate*). También debemos efectuar una Transformación de datos desde el formato del Mapa de registros y un mensaje de inserción.

8.2.2. Creación de una transformación de datos

Encontraremos el Generador para la Transformación de datos (DT) en el menú [Interoperability > Builder]. A continuación, crearemos nuestra DT de esta forma (si no encuentras la clase Formation.RM.Csv.Record, tal vez no se generó el Mapa de Registros):

Ahora, podemos mapear los diferentes campos juntos:

8.2.3. Añadir la transformación de datos al *business process*

Lo primero que debemos modificar es la clase de la solicitud del *business process*, ya que necesitamos tener una entrada en el Mapa de Registros que creamos.

Entonces, podemos añadir nuestra transformación (el nombre del proceso no cambia nada, a partir de aquí elegimos llamarlo Main):

La actividad de transformación tomará la solicitud del *business process* (un registro del archivo CSV, gracias a nuestro servicio para mapear los registros) y la transformará en un mensaje FormationInsertRequest. Si deseamos almacenar ese mensaje para enviarlo al *business operation*, necesitamos añadir una propiedad al contexto del *business process*.

Ahora podemos configurar nuestra función de transformación para que tome su entrada como entrada del *business process* y guarde su salida en la propiedad recién creada. El origen y el objetivo de la transformación RmToMsg son request y context.Msg, respectivamente:

Tenemos que hacer lo mismo para Call BO. Su entrada, o callrequest, es el valor almacenado en context.msg:

Al final, el flujo en el *business process* puede representarse de esta manera:

8.2.4. Configuración de la producción

Con el signo [+], podemos añadir nuestro nuevo proceso a la producción (si aún no lo hemos hecho). También necesitamos un servicio genérico para utilizar el Mapa de Registros, para ello utilizamos

EnsLib.RecordMap.Service.FileService (lo añadimos con el botón [+] que está junto a los servicios). A continuación, parametrizamos este servicio:

Ahora deberíamos poder probar nuestro *business process*.

8.2.5. Pruebas

Probamos toda la producción de esta manera:

En el menú System Explorer > SQL, puedes ejecutar el comando

```
SELECT
ID, Name, Salle
FROM Formation_Table.Formation
```

para ver los objetos que acabamos de guardar.

9. Obtener acceso a una base de datos externa usando JDBC

En esta sección, crearemos una operación para guardar nuestros objetos en una base de datos externa. Utilizaremos la API de JDBC, así como el otro contenedor Docker que configuramos, con postgre en él.

9.1. Creación de nuestra nueva operación

Nuestra nueva operación, en el archivo Formation/BO/RemoteBDD.cls es así:

```
Include EnsSQLTypes

Class Formation.BO.RemoteBDD Extends Ens.BusinessOperation
{

Parameter ADAPTER = "EnsLib.SQL.OutboundAdapter";

Property Adapter As EnsLib.SQL.OutboundAdapter;

Parameter INVOCATION = "Queue";

Method InsertRemoteBDD(pRequest As Formation.Msg.FormationInsertRequest, Output pResp
onse As Ens.StringResponse) As %Status
{
    set tStatus = $$$OK

    try{
        set pResponse = ##class(Ens.Response).%New()
        set ^inc = $I(^inc)
        set tInsertSql = "INSERT INTO public.formation (id, nom, salle) VALUES(?, ?,
?)"
        $$$ThrowOnError(..Adapter.ExecuteUpdate(.nrows,tInsertSql,^inc,pRequest.Forma
tion.Nom, pRequest.Formation.Salle ))
    }
    catch exp
    {
        Set tStatus = exp.AsStatus()
```

```
    }

    Quit tStatus
}

XData MessageMap
{
<MapItems>
    <MapItem MessageType="Formation.Msg.FormationInsertRequest">
        <Method>InsertRemoteBDD</Method>
    </MapItem>
</MapItems>
}

}
```

Esta operación es similar a la primera que creamos. Cuando reciba un mensaje del tipo Formation.Msg.FormationInsertRequest, utilizará un adaptador para ejecutar solicitudes SQL. Esas solicitudes se enviarán a nuestra base de datos de postgre.

9.2. Configuración de la producción

Ahora, desde el Portal de Administración, crearemos una instancia de esa operación (añadiéndola con el signo [+] en la producción).

También necesitaremos añadir el JavaGateway para el controlador JDBC en los servicios. El nombre completo de este servicio es EnsLib.JavaGateway.Service.

Ahora necesitamos configurar nuestra operación. Como hemos configurado un contenedor postgre, y conectado su puerto 5432, los valores que necesitamos en los siguientes parámetros son:

DSN: jdbc:postgresql://db:5432/DemoData

Controlador de JDBC: org.postgresql.Driver

JDBC Classpath: /tmp/iris/postgresql-42.2.14.jar

Finalmente, necesitamos configurar las credenciales para tener acceso a la base de datos remota. Para ello, necesitamos abrir el Visualizador de credenciales:

Tanto el nombre de usuario como la contraseña son DemoData, como lo configuramos en el archivo docker-compose.yml.

De vuelta a la producción, podemos añadir "Postgre" en el campo [Credential] en la configuración de nuestra operación (debería estar en el menú desplegable). Antes de que podamos probarlo, debemos añadir el JGService a la operación. En la pestaña [Settings], en [Additional Settings]:

9.3. Pruebas

Cuando se están haciendo pruebas el registro visual debe mostrar que tuvimos éxito:

Conectamos correctamente una base de datos externa.

9.4. Ejercicio

Como ejercicio, podría ser interesante modificar BO.LocalBDD para que devuelva un booleano, que le dirá al *business process* que llame a BO.RemoteBDD dependiendo del valor de ese booleano.

Sugerencia: Esto se puede hacer cambiando el tipo de respuesta que devuelve LocalBDD, añadiendo una nueva propiedad al contexto y utilizando la actividad if en nuestro *business process*.

9.5. Solución

Primero, necesitamos tener una respuesta de nuestra operación LocalBDD. Vamos a crear un nuevo mensaje, en Formation/Msg/FormationInsertResponse.cls:

```
Class Formation.Msg.FormationInsertResponse Extends Ens.Response
{

Property Double As %Boolean;

}
```

Después, cambiamos la respuesta de LocalBDD por esa respuesta y establecemos el valor de su booleano de forma aleatoria (o no):

```
Method InsertLocalBDD(pRequest As Formation.Msg.FormationInsertRequest, Output pRespo
nse As Formation.Msg.FormationInsertResponse) As %Status
{
    set tStatus = $$$OK

    try{
        set pResponse = ##class(Formation.Msg.FormationInsertResponse).%New()
        if $RANDOM(10) < 5 {
            set pResponse.Double = 1
        }
        else {
            set pResponse.Double = 0
        }
...
```

Ahora crearemos un nuevo proceso (copiado del que hicimos), donde añadiremos una nueva propiedad de contexto, de tipo %Boolean:

Esta propiedad se completará con el valor del callresponse.Double de nuestra llamada de operación (necesitaremos establecer [Response Message Class] en nuestra nueva clase de mensaje):

A continuación, añadimos una actividad if, con la propiedad context.Double como condición:

MUY IMPORTANTE: necesitamos desmarcar **Asynchronous** en la configuración de nuestra LocallBDD Call, o la actividad if se activará antes de que reciba la respuesta booleana.

Finalmente configuramos nuestra actividad de llamada como un objetivo RemoteBDD BO:

Para completar la actividad if, necesitamos arrastrar otro conector desde la salida del if al triángulo join que se encuentra debajo. Como no haremos nada si el valor booleano es falso, dejaremos este conector vacío.

Después de compilar y crear instancias, deberíamos poder probar nuestro nuevo proceso. Para ello, necesitamos cambiar Target Config Name de nuestro servicio de archivos.

En el seguimiento, deberíamos tener aproximadamente la mitad de los objetos leídos en el csv guardados también en la base de datos remota.

10. Servicio REST

En esta parte, crearemos y utilizaremos un Servicio REST.

10.1. Creación del servicio

Para crear un servicio REST, necesitamos una clase que extienda %CSP.REST, en Formation/REST/Dispatch.cls tenemos:

```
Class Formation.REST.Dispatch Extends %CSP.REST
{

/// Ignore any writes done directly by the REST method.
Parameter IgnoreWrites = 0;

/// By default convert the input stream to Unicode
Parameter CONVERTINPUTSTREAM = 1;

/// The default response charset is utf-8
Parameter CHARSET = "utf-8";

Parameter HandleCorsRequest = 1;

XData UrlMap [ XMLNamespace = "http://www.intersystems.com/urlmap" ]
{
<Routes>
  <!-- Get this spec -->
  <Route Url="/import" Method="post" Call="import" />
</Routes>
}

/// Get this spec
ClassMethod import() As %Status
{
  set tSc = $$$OK

  Try {

      set tBsName = "Formation.BS.RestInput"
      set tMsg = ##class(Formation.Msg.FormationInsertRequest).%New()

      set body = $zcvt(%request.Content.Read(),"I","UTF8")
      set dyna = {}.%FromJSON(body)

      set tFormation = ##class(Formation.Obj.Formation).%New()
      set tFormation.Nom = dyna.nom
      set tFormation.Salle = dyna.salle

      set tMsg.Formation = tFormation
```

```
      $$$ThrowOnError(##class(Ens.Director).CreateBusinessService(tBsName,.tService))

      $$$ThrowOnError(tService.ProcessInput(tMsg,.output))

  } Catch ex {
      set tSc = ex.AsStatus()
  }

  Quit tSc
}

}
```

Esta clase contiene una ruta para importar un objeto, vinculado al verbo POST:

```
<Routes>
  <!-- Get this spec -->
  <Route Url="/import" Method="post" Call="import" />
</Routes>
```

El método de importación creará un mensaje que se enviará a un *business service*.

10.2. Añadir nuestro *Business Service* (BS)

Vamos a crear una clase genérica que dirigirá todas sus solicitudes hacia TargetConfigNames. Este objetivo se configurará cuando creemos una instancia de este servicio. En el archivo Formation/BS/RestInput.cls tenemos:

```
Class Formation.BS.RestInput Extends Ens.BusinessService
{

Property TargetConfigNames As %String(MAXLEN = 1000) [ InitialExpression = "BuisnessP
rocess" ];

Parameter SETTINGS = "TargetConfigNames:Basic:selector?multiSelect=1&context={Ens.Con
textSearch/ProductionItems?targets=1&productionName=@productionId}";

Method OnProcessInput(pDocIn As %RegisteredObject, Output pDocOut As %RegisteredObjec
t) As %Status
{
    set status = $$$OK

    try {

        for iTarget=1:1:$L(..TargetConfigNames, ",") {
            set tOneTarget=$ZStrip($P(..TargetConfigNames,",",iTarget),"<>W")  Contin
ue:""=tOneTarget
            $$$ThrowOnError(..SendRequestSync(tOneTarget,pDocIn,.pDocOut))
        }
    } catch ex {
        set status = ex.AsStatus()
    }

    Quit status
```

```
}

}
```

Volviendo a la configuración de la producción, añadimos el servicio de la manera habitual. En [Target Config Names], ponemos nuestro BO LocalBDD:

Para utilizar este servicio, necesitamos publicarlo. Para ello, usamos el menú [Edit Web Application]:

10.3. Pruebas

Por último, podemos probar nuestro servicio con cualquier tipo de cliente REST:

Conclusión

A través de esta formación, hemos creado una producción que es capaz de leer líneas desde un archivo csv y guardar los datos leídos tanto en la base de datos de IRIS como en una base de datos externa, utilizando JDBC. También añadimos un servicio REST para utilizar el verbo POST para guardar nuevos objetos.

Hemos descubierto los principales elementos del *framework* de interoperabilidad de InterSystems.

Lo hemos hecho utilizando Docker, VSCode y el Portal de Administración de InterSystems IRIS.

#Docker #Framework #Interoperabilidad #Principiante #Caché #Ensemble #InterSystems IRIS
Ir a la aplicación en InterSystems Open Exchange

URL de fuente:https://es.community.intersystems.com/post/curso-de-formaci%C3%B3n-sobre-ensemble-interoperabilidad