# Uso del Hub

Instrucciones de uso y configuración

## Introducción

El Hub es un producto que permite integrar diversas API externas con CIPE API, permitiendo configurar transacciones con modelos de autenticación, parámetros de entrada y un mapa de resultados.

La respuesta del Hub tiene un formato normalizado, haciendo que los distintos tipos de respuesta se traduzca a una estructura normalizada, sin sorpresas.

## Estructura del Proyecto

* HUB
  + Controllers
    - IntegrationController.cs
  + Integrations
    - api\_fiscalia
      * v1
        + api\_fiscalia.cs
        + api\_fiscalia.yml
    - BaseIntegrationApi.cs
    - IIntegrationApi.cs
  + Models
    - ApiConfiguracion.cs
    - OutputCampo.cs
    - ParametroDefinicion.cs
    - TransaccionDefinicion.cs
  + Services
    - IntegrationService.cs
  + Transactions
    - secure
      * personas
        + list.yml
    - paises.yml
    - personas.yml
  + Utils
    - ParamValidators.cs
* Controllers/**IntegrationController.cs**: Controlador único del Hub, diseñado para capturar todas las llamadas que apunten al Hub y resolverlas de acuerdo a la configuración que corresponda:
  + api/<versión>/<endpoint>
* Integrations: Contiene la lógica que permite realizar llamados a APIs externas, además de la configuración de cada una de las API externas a utilizar por las transacciones, estas últimas van siempre en carpetas con su nombre:
  + api\_fiscalia/v1: Estas carpetas definen el nombre de la API y la versión
    - api\_fiscalia.cs: Declaración de una API externa, no contiene código fuente específico ya que realiza llamadas estándar.
    - api\_fiscalia.yml: Configuración base de la API externa, determina la URL destino, el tiempo máximo de espera, el tipo de respuesta, los headers y el modelo de autenticación a utilizar.
  + **BaseIntegrationApi.cs**: Hace transporte/autenticación, carga la configuración de la integración, prepara HttpClient, aplica auth, arma query/body y ejecuta.
  + IIntegrationApi.cs: Interfaz de integración, permite definir la estructura mínima con que debe contar una integración para poder heredar BaseIntegrationApi
* Models: Modelos de estandarización de datos
  + ApiConfiguracion.cs
  + OutputCampo.cs
  + ParametroDefinicion.cs
  + TransaccionDefinicion.cs
* Services/**IntegrationService.cs**: Orquestador, resuelve la transacción YAML, valida inputs, arma path params, llama a la API destino y mapea la salida.
* Utils/ParamValidators.cs: Contiene validadores de parámetros (entradas) para robustecer el consumo de transacciones, es la primera capa de validación para no ensuciar las APIs destino con valores incorrectos.

## Conceptos Clave

1. El Hub es una API, por lo que escucha peticiones en una URL con un endpoint de tipo Post
2. Los parámetros para llamar al Hub se envían en formato JSON, siempre en el body de la operación, en la configuración de cada transacción se define si se usarán en el query, path o body de la llamada al servicio externo, de ese modo las llamadas quedan 100% normalizadas.
3. El Hub responde los resultados en una estructura JSON estandarizada, sin importar que la API externa devuelva los valores en formatos propietarios, el formato incluye los siguientes datos:
   1. status: Si es ‘success’ la ejecución se realizó con éxito
   2. message: Contiene un texto relacionado con el estado de la llamada
   3. output: Contiene una lista con el (o los) resultado(s)
   4. time: Contiene los tiempos de ejecución, separados en ‘remote’, ‘hub’ y ‘total’, correspondientes al tiempo consumido por la API externa, el tiempo consumido por el Hub, y la suma de ambos respectivamente.
4. Cada API externa se declara en la carpeta Integrations, luego, se pueden crear múltiples transacciones del tipo declarado escribiendo un archivo YAML, que no requiere programación.
5. No es necesario crear controladores, el Hub opera de forma integral, traduciendo cualquier endpoint a una ruta interna, la que contendrá la definición de la transacción solicitada.
6. La API Cipe fue modificada para soportar transacciones de tipo HubConnector, permitiendo consumir el Hub de manera transparente, guardando logs de auditoría incluyendo los timings para detectar cualquier problema de performance que no sea responsabilidad de ACSIM.

## Creación de una integración:

1. Crea una carpeta en Integrations con el nombre técnico de la API Externa, sin espacios, por ejemplo: api\_fiscalia
   1. Integrations/api\_fiscalia/
2. Crea una clase con el mismo nombre de la carpeta
   1. Integrations/api\_fiscalia/api\_fiscalia.cs
3. Edita la clase y pega el siguiente contenido:

namespace Hub.Integrations.api\_fiscalia.v1;

public class ApiFiscalia : BaseIntegrationApi

{

public ApiFiscalia(IHttpClientFactory factory) : base(factory) { }

protected override string ApiName => "api\_fiscalia";

protected override string ApiVersion => "v1";

}

1. Asegúrate de usar el nombre técnico en el namespace

namespace Hub.Integrations.**api\_fiscalia**.v1;

1. Asigna un nombre PascalCase a la clase y a su constructor:

ApiFiscalia (o el que estés creando)

1. Declara ApiName con el nombre técnico y ApiVersion con v1 (según corresponda)

protected override string ApiName => "**api\_fiscalia**";

protected override string ApiVersion => "**v1**";

1. Crea un archivo YAML con el mismo nombre de la carpeta

Integrations/api\_fiscalia/api\_fiscalia.yml

1. Edita el archivo y pega el siguiente contenido:

url: https://localhost:5003/api/v1

timeout: 15

body\_type: json

headers:

Content-Type: application/json

auth:

type: none

1. Modifica el valor de url indicando la ruta base de la API que quieres integrar, en este ejemplo estamos usando una ruta a una api mock local.
2. Si la API externa usa algún tipo de autenticación específico, reemplaza ‘none’ (ninguno) por uno de los siguientes tipos:
   1. **basic**: Si la API sólo requiere usuario y clave
   2. **oauth2**: Si la API usa el modelo oAuth de Google
   3. **aws\_sigv4**: Si la API usa el modelo SigV4 de Amazon Web Services
   4. **bearer**: Si la API usa un Token fijo
   5. **api\_key\_header**: Si la API espera una llave personalizada en el header
   6. **mtls**: Si la API usa este esquema de alta seguridad (corp, banca, etc.)
   7. **ntlm**: Si la API usa seguridad Windows/Active Directory/Sharepoint
   8. **hmac\_sha256**: Si la API requiere firma HMAC para garantizar integridad y autenticidad sin depender de un emisor oAuth

## Creación de una transacción:

Supongamos que existe un endpoint llamado ‘paises’ en la API de Fiscalía, que permite buscar un país del path, ejemplo: /paises/chile o /países/argentina

1. Elije un endpoint apropiado para tu nueva transacción, ejemplo:

**geo/pais**

1. No es necesario replicar el endpoint de la API externa, el Hub te permite agrupar por funcionalidad o el criterio que quieras, así que le añadimos una ruta previa llamada ‘geo’, que en el futuro contendrá endpoints relacionados, aunque no se resuelvan con las mismas APIs externas.
2. Dado que creamos un endpoint con ruta (geo), asegúrate de replicar la ruta en la carpeta Transactions, creando el archivo pais.yml:

**Transactions\geo\pais.yml**

**Nota**: El Hub permite que, en lugar de usar carpetas, crees transacciones en 1 archivo en la raíz, con las rutas separadas por punto, en este caso:

Transactions\geo.pais.yml (nótese el punto separador del nombre)

1. Edita pais.yml y pega el siguiente contenido:

api:

name: api\_fiscalia

version: v1

endpoint: pais/{pais}

method: get

input:

- name: pais

type: string

required: true

location: path

output:

- name: nombre

allow\_filter: false

- name: capital

allow\_filter: false

- name: poblacion\_millones

allow\_filter: false

1. A continuación, una explicación del YAML de transacción:
   1. **api**: Configuración de la llamada
      * name: Nombre de la integración a usar
      * version: Versión de la API destino, normalmente v1
      * endpoint: Ruta del endpoint, sin incluir la ruta base
      * method: Método que se usará en esta transacción (get, post)
   2. **input**: Parámetros de entrada (arreglo)
      * - name: Nombre del parámetro
      * type: Tipo de dato que debe recibirse, ejemplo: string
      * required: true si es obligatorio
      * **location**: Donde irá el parámetro (query, path o body)
   3. **output**: Mapeo de los resultados
      * – name: <nombre del campo en la respuesta>
      * allow\_filter: true o false, si es que quieres o no filtrar el resultado por este campo (subset de datos)
      * alias: Permite renombrar el campo a tu gusto (opcional)

**Notas de location:**

* Los parámetros tipo ‘path’ deben incluirse en el endpoint de la transacción en la posición que corresponda, entre llaves, ejemplo: endpoint: pais/{pais}
* Los parámetros tipo ‘query’ no deben ser incluidos en el endpoint, ya que el Hub los añadirá automáticamente.

## Probando la transacción

1. Desde VSCode, inicia el Hub teniendo tu nueva transacción ya configurada, asegúrate de que la API externa esté activa y respondiendo, si no tienes dicha API, utiliza la ApiMock incluida con el Hub, emulando su funcionamiento.
2. Por defecto el Hub opera localmente en: [**https://localhost:5000/api/v1**](https://localhost:5000/api/v1)
3. En Postman, crea un request (Post) apuntando a: [**https://localhost:5000/api/v1/pais**](https://localhost:5000/api/v1/pais)
4. En el Body (raw/JSON) entrega el parámetro de entrada ‘pais’:

{

"pais": "chile"

}

1. Ejecuta y si todo está correcto, recibirás una respuesta similar a ésta:

{

"status": "success",

"message": "Operación realizada con éxito",

"output": [

{

"nombre": "Chile",

"capital": "Santiago",

"poblacion\_millones": 19.5

}

],

"time": {

"total": 0.1124,

"remote": 0.0893,

"hub": 0.0231

}

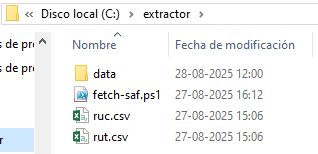
}

1. El resultado contendrá el estado, la información y la medición.

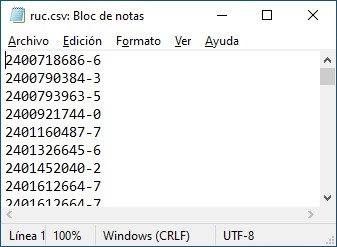
## Script PowerShell para extracción de datos

En conjunto con el Hub y la ApiMock, existe un script PoweShell que ejecuta las 3 API de la fiscalía leyendo RUT y RUC desde archivos CSV y generando archivos JSON con los resultados, en un formato compatible con un programa que los procesa y genera una planilla Excel con la información formateada.

1. Ubicación del script: C:\extractor
2. Nombre del script: fetch-saf.ps1



1. Los archivos ruc.csv y rut.csv deben contener 1 valor por línea, sin comillas ni separadores, tal como a continuación:



1. Para iniciar la captura, se debe pulsar con el botón derecho del mouse, sobre fetch-saf.ps1 y elegir la opción “Ejecutar con Powershell”, tal como en la imagen:

Imagen que contiene Interfaz de usuario gráfica

El contenido generado por IA puede ser incorrecto.

1. El proceso abrirá una ventana de tipo consola y mostrará el avance, puede tardar varios minutos dependiendo de la cantidad de rut o ruc definidos en cada archivo csv. Tener en cuenta que ciertos clientes pueden tener mucho historial, llegando a superar los 140mb, lo que impactará en el tiempo de proceso.
2. Al finalizar, el proceso generará una carpeta llamada data con cada resultado en un archivo JSON, tener en cuenta que la API de Fiscalía puede fallar, en cuyo caso se generará un archivo con nombre terminado en .error, ejemplo:

persona.info.0000017543228-0.error.txt

1. Una vez generados los archivos, comprimirlos y copiarlos en una máquina local para procesarlos con el programa MinPub JSON Converter, explicado en el siguiente apartado.

## MinPub JSON Converter

Con el fin de procesar los archivos JSON generados con el Shell de extracción de datos, se creó un conversor (aplicación de escritorio) que leerá los archivos JSON que existan en una carpeta, permitiendo crear una planilla Excel con la información integrada.

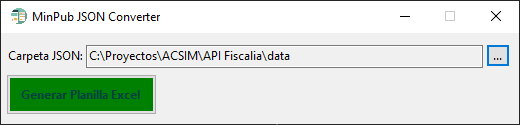
1. **Conversor**: El conversor, cuyo instalador se llama MinPubJSONConverter-Setup-1.0.0.exe, debe ser instalado en la máquina local para poder procesar los archivos JSON:



1. **Archivos JSON**: Dejar los archivos JSON obtenidos con el Shell descrito en el apartado anterior en una carpeta local, por ejemplo, en el escritorio, siempre en una carpeta donde no se mezcle con otros archivos, ejemplo:

Escritorio\Datos20250901

1. **Proceso**: Abrir el programa y seleccionar la carpeta con los archivos JSON:



1. Pulsar el botón “Generar Planilla Excel”, dar el nombre y ubicación de la planilla a generar y luego esperar hasta que aparezca el aviso de que el proceso concluyó, abra la planilla generada para ver los resultados.