Article
Guillaume Rongier · Juin 15, 2022 17m de lecture

Open Exchange

Comment publier rapidement des RESTful API dans OAS 3.0 à l'aide de l'outil IRIS ApiPub

Introduction

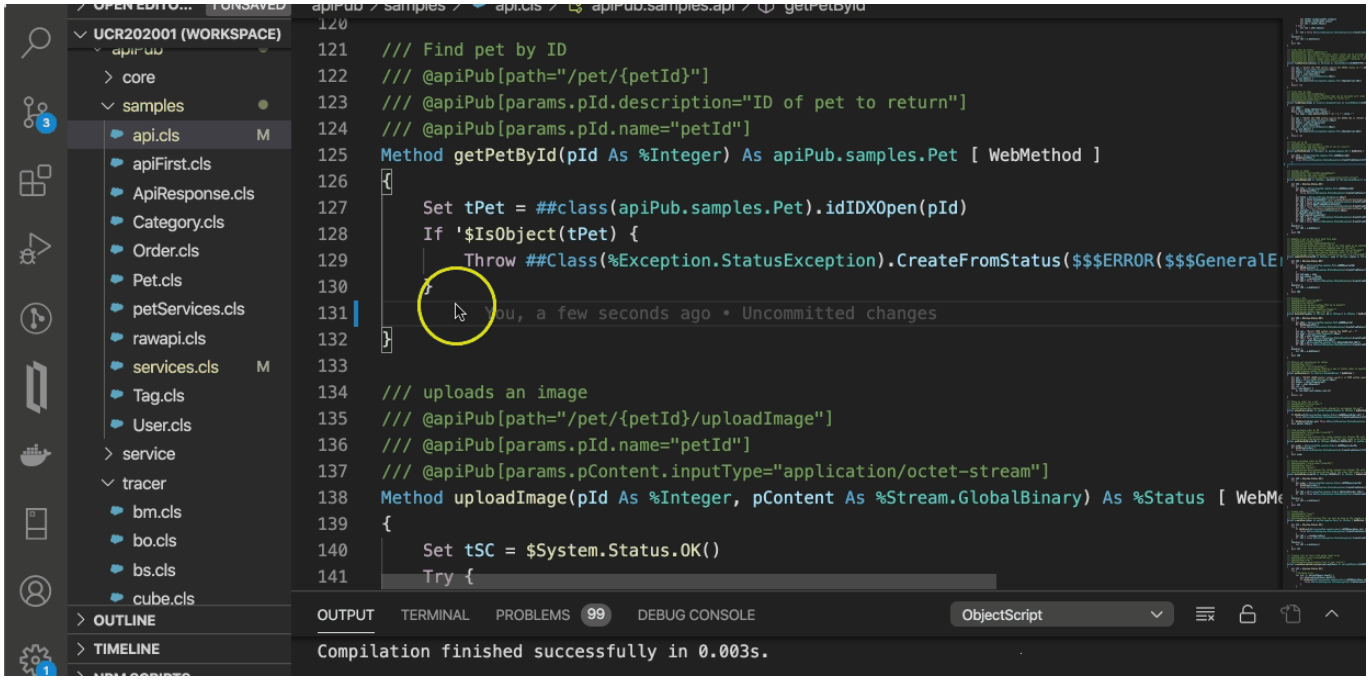
Nous sommes à l'ère de **l'économie multiplateforme** et les API sont la *"colle "* de ce scénario numérique. Étant donné leur importance, les développeurs les considèrent comme un **service** ou un **produit** à consommer. Par conséquent, **l'expérience** d'utilisation est un facteur crucial de leur **succès**.

Afin d'améliorer cette expérience, des normes de spécification telles que la spécification OpenAPI (OAS) sont de plus en plus adoptées dans le développement des API RESTFul.

IRIS ApiPub - qu'est-ce que c'est ?

IRIS ApiPub est un projet de type code source ouvert Open Source dont l'objectif principal est de **publier** automatiquement **les API RESTful** créées avec la technologie Intersystems IRIS, de la manière la plus simple et la plus rapide possible en utilisant la norme Open Specification API (OAS) standard, version 3.0.

Il permet à l'utilisateur de se concentrer sur la **mise en œuvre** et les **règles métier** (méthodes Web) de l'API, en abstrayant et en automatisant les autres aspects liés à **la documentation, l'exposition, l'exécution et la surveillance** des services.



Ce projet comprend également un exemple complet de mise en œuvre (**apiPub.samples.api**) de la Swagger Petstore, qui est l'*échantillon* officiel de swagger.

Testez-le avec vos services SOAP actuels

Si vous avez déjà publié des services SOAP, vous pouvez les tester en utilisant Rest/JSON avec OAS 3.0.

Lors de la publication de méthodes avec des types complexes, la classe de l'objet doit être une sous-classe de %XML.Adapter. De cette manière, les services SOAP précédemment installés sont rendus automatiquement compatibles.

Surveillez vos API avec IRIS Analytics

Activez la surveillance des API pour **gérer** et **suivre** tous vos *Appels Rest*. Vous pouvez également configurer vos propres indicateurs.

Installation

1. Effectuez un *clone/git pull* depuis le dépôt dans le répertoire local.

```
$ git clone https://github.com/devecchijr/apiPub.git
```

2. Ouvrez le terminal dans ce répertoire et exécutez la commande suivante :

```
$ docker-compose up -d
```

3. Exécutez le conteneur IRIS avec le projet :

```
$ docker-compose up -d
```

Test de l'application

Ouvrez l'URL http://localhost:52773/swagger-ui/index.html de swagger

Essayez d'exécuter une opération à l'aide de l'API Petstore, par exemple en effectuant un *postage* d'un nouveau *pet*.

Consultez la table de bord du moniteur apiPub. Essayez d'explorer le domaine petStore pour explorer et analyser les messages.

Modifiez ou créez des méthodes dans la classe apiPub.samples.api et revenez à la documentation générée. Notez que toutes les modifications sont automatiquement reflétées dans la documentation OAS ou dans les schémas.

Publiez votre API selon la norme OAS 3.0 en seulement 3 étapes :

Étape 1

Définissez la classe d'implémentation de votre API et **balises** les méthodes avec l'attribut [WebMethod]

Cette étape n'est pas nécessaire si vous disposez déjà d'une mise en œuvre de WebServices.

Étape 2

Créez une **sous-classe** de apiPub.core.service et définissez sa propriété DispatchClass comme la classe Implementation créée précédemment. Incluez également le chemin de la documentation OAS 3.0. Si vous le souhaitez, pointez vers la classe apiPub.samples.api (PetStore).

Étape 3

Créez une application Web et définissez la classe de répartition comme la classe de service créée ci-dessus.

Utilisation de Swagger

Avec iris-web-swagger-ui vous pouvez exposer votre spécification de service. Il vous suffit de pointer vers le chemin de la documentation et... **VOILÁ!!**

Définition de l'en-tête de la spécification OAS

Il existe deux façons de définir l'en-tête OAS 3.0 :

La première consiste à créer un bloc JSON XDATA nommé *apiPub* dans la classe d'implémentation. Cette méthode autorise plus d'une balise et la modélisation est compatible avec la norme OAS 3.0. Les propriétés qui peuvent être personnalisées sont *info, tags* et *servers*.

```
XData apiPub [ MimeType = application/json ]
{
    {
        "info" : {
            "description" : "Il s'agit d'un exemple de serveur Petstore.  Vous pouvez
 en savoir plus sur Swagger à l'adresse suivante\n[http://swagger.io](http://swagger.
io) or on\n[irc.freenode.net, #swagger](http://swagger.io/irc/).\n",
            "version" : "1.0.0",
            "title" : "IRIS Petstore (Dev First)",
            "termsOfService" : "http://swagger.io/terms/",
            "contact" : {
            "email" : "apiteam@swagger.io"
            },
            "license" : {
            "name" : "Apache 2.0",
            "url" : "http://www.apache.org/licenses/LICENSE-2.0.html"
            }
        },
        "tags" : [ {
            "name" : "pet",
            "description" : "Tout sur vos Pets",
            "externalDocs" : {
            "description" : "Pour en savoir plus",
            "url" : "http://swagger.io"
            }
        }, {
            "name" : "store",
            "description" : "Accès aux commandes du Petstore"
        }, {
            "name" : "user",
            "description" : "Opérations sur l'utilisateur",
            "externalDocs" : {
            "description" : "En savoir plus sur notre magasin",
            "url" : "http://swagger.io"
            }
        } ]
```

```
    }
}
```

La seconde méthode consiste à définir des paramètres dans la classe d'implémentation, comme dans l'exemple suivant :

```
Parameter SERVICENAME = "My Service";

Parameter SERVICEURL = "http://localhost:52776/apipub";

Parameter TITLE As %String = "REST aux API SOAP";

Parameter DESCRIPTION As %String = "API pour le proxy des services Web SOAP via REST"
;

Parameter TERMSOFSERVICE As %String = "http://www.intersystems.com/terms-of-
service/";

Parameter CONTACTNAME As %String = "John Doe";

Parameter CONTACTURL As %String = "https://www.intersystems.com/who-we-are/contact-
us/";

Parameter CONTACTEMAIL As %String = "support@intersystems.com";

Parameter LICENSENAME As %String = "Copyright InterSystems Corporation, tous droits r
éservés.";

Parameter LICENSEURL As %String = "http://docs.intersystems.com/latest/csp/docbook/co
pyright.pdf";

Parameter VERSION As %String = "1.0.0";

Parameter TAGNAME As %String = "Services";

Parameter TAGDESCRIPTION As %String = "Services d'héritage";

Parameter TAGDOCSDESCRIPTION As %String = "Pour en savoir plus";

Parameter TAGDOCSURL As %String = "http://intersystems.com";
```

Personnalisez vos API

Vous pouvez personnaliser plusieurs aspects de l'API, tels que les ***balises, les chemins et les verbes***. Pour cela, vous devez utiliser une notation spéciale, déclarée dans le commentaire de la méthode personnalisée.

Syntaxe:

```
/// @apiPub[assignment clause]
  [Method/ClassMethod] methodName(params as type) As returnType {

}
```

Toutes les personnalisations présentées à titre d'exemple dans cette documentation se trouvent dans la classe

apiPub.samples.api.

Personnalisation des verbes

Lorsqu'aucun type complexe n'est utilisé comme paramètre d'entrée, apiPub attribue automatiquement le verbe *Get*. Dans le cas contraire, le verbe *Post* sera attribué.

Si vous souhaitez personnaliser la méthode, ajoutez la ligne suivante aux commentaires de la méthode.

```
/// @apiPub[verb="verb"]
```

Où *verbe* peut être **get, post, put, delete ou patch**.

Exemple:

```
/// @apiPub[verb="put"]
```

Personnalisation des chemins

Cet outil attribue automatiquement des *chemins* ou des routages aux *Méthodes Web*. Il utilise le nom de la méthode comme *chemin*, par défaut.

Si vous souhaitez personnaliser le **chemin**, ajoutez la ligne suivante aux commentaires de la méthode.

```
/// @apiPub[path="path"]
```

Où *chemin* peut être n'importe quelle valeur précédée d'un slash, tant qu'il n'entre pas en conflit avec un autre *chemin* dans la même classe d'implémentation.

Exemple:

```
/// @apiPub[path="/pet"]
```

Une autre utilisation très courante du chemin est de définir un ou plusieurs paramètres dans le chemin lui-même. Pour cela, le nom du paramètre défini dans la méthode doit être entouré d'accolades.

Exemple:

```
/// @apiPub[path="/pet/{petId}"]
  Method getPetById(petId As %Integer) As apiPub.samples.Pet [ WebMethod ]
  {
  }
```

Lorsque le nom du paramètre interne diffère du nom du paramètre affiché, le nom peut être égalisé selon l'exemple suivant :

```
/// @apiPub[path="/pet/{petId}"]
```

```
/// @apiPub[params.pId.name="petId"]
Method getPetById(pId As %Integer) As apiPub.samples.Pet [ WebMethod ]
{
}
```

Dans l'exemple ci-dessus, le paramètre interne *pId* est affiché sous la forme *petId*.

Personnalisation des balises

Il est possible de définir le **tag** (regroupement) de la méthode lorsque plus d'un tag est défini dans l'en-tête.

```
/// @apiPub[tag="value"]
```

Exemple:

```
/// @apiPub[tag="user"]
```

Personnalisation du succès *Code d'état*

Si vous souhaitez modifier le *Code d'état* de réussite de la méthode, qui est ***200*** par défaut, il convient d'utiliser la notation suivante.

```
/// @apiPub[successfulCode="code"]
```

Exemple:

```
/// @apiPub[successfulCode="201"]
```

Personnalisation de l'exception *Code d'état*

Cet outil traite toutes les exceptions comme ***Code d'état 500*** par défaut. Si vous souhaitez ajouter de nouveaux codes d'exception à la documentation, utilisez la notation suivante.

```
/// @apiPub[statusCodes=[{code:"code",description:"description"}]]
```

Où la propriété *statusCodes* est un tableau d'objets contenant le code et la description.

Exemple:

```
/// @apiPub[statusCodes=[
  /// {"code":"400","description":"Invalid ID supplied"}
  /// ,{"code":"404","description":"Pet not found"}]
  /// ]
```

Lorsque vous soulevez l'exception, incluez *Code d'état* dans la description de l'exception entre les caractères "<" et ">".

Exemple:

```
Throw ##Class(%Exception.StatusException).CreateFromStatus($$$ERROR($$$GeneralError, " Invalid ID supplied"))}
```

Voir la méthode ***getPetById*** de la classe apiPub.samples.api

Marquer l'API comme déconseillé

Pour que l'API soit affichée comme ***déconseillée***, la notation suivante doit être utilisée :

```
/// @apiPub[deprecated="true"]
```

Personnalisation de l'*operationId*

Selon la spécification OAS, ***operationId*** est une chaîne unique utilisée pour identifier une API ou une opération. Dans cet outil, il est utilisé dans le même but lors des opérations de surveillance et suivi operations.

Par défaut, elle porte le même nom que la méthode de la classe d'implémentation.

Si vous souhaitez le modifier, utilisez la notation suivante

```
/// @apiPub[operationId="updatePetWithForm"]
```

Modification du jeu de caractères de la méthode

Le jeu de caractères par défaut est généralement défini à l'aide du paramètre CHARSET de la classe de service, décrit dans étape 2. Si vous souhaitez personnaliser le jeu de caractères d'une méthode, vous devez utiliser la notation suivante ::

```
/// @apiPub[charset="value"]
```

Exemple:

```
/// @apiPub[charset="UTF-8"]
```

Personnalisation des noms et autres caractéristiques des paramètres

Vous pouvez personnaliser plusieurs aspects des paramètres d'entrée et de sortie de chaque méthode, tels que les noms et les descriptions qui seront affichés pour chaque paramètre.

Pour personnaliser un paramètre spécifique, utilisez la notation suivante

```
/// @apiPub[params.paramId.property="value"]
```

ou pour des réponses :

```
/// @apiPub[response.property="value"]
```

Exemple:

```
/// @apiPub[params.pId.name="petId"]
 /// @apiPub[params.pId.description="ID of pet to return"]
```

Dans ce cas, le nom *petId* et la description *ID du pet à rendre* sont attribués au paramètre défini comme *pId*

Lorsque la personnalisation n'est pas spécifique à un paramètre donné, la notation suivante est utilisée

```
/// @apiPub[params.property="value"]
```

Dans l'exemple suivant, la description *Ceci ne peut être fait que par l'utilisateur connecté* est attribuée à l'ensemble de la *demande*, et pas seulement à un seul paramètre :

```
/// @apiPub[params.description="Ceci ne peut être fait que par l'utilisateur connecté."]
```

Autres propriétés qui peuvent être personnalisées pour des paramètres spécifiques

Utilisez la notation suivante pour les paramètres d'entrée ou de sortie :

```
/// @apiPub[params.paramId.property="value"]
```

Pour les reponses:

```
/// @apiPub[response.property="value"]
```

Propriété
required: "true" si le paramètre est obligatoire. Tous les paramètres de type **path** sont déjà automatiquement requis
schema.items.enum: afficher les énumérateurs pour les types %String ou %Library.DynamicArray. Voir la méthode ***findByStatus*** de la classe apiPub.samples.api
schema.default: Pointe vers une valeur par défaut pour les énumérateurs
inputType: Pour les types simples, il s'agit par défaut d'un **paramètre de requête**. Pour les types complexes (corps), il s'agit par défaut de **application/json**. Dans le cas où vous souhaitez changer le type d'entrée, vous pouvez utiliser ce paramètre. Exemple d'utilisation : Téléchargement d'une image, qui n'est généralement pas de type JSON. Voir la méthode ***uploadImage*** de la classe apiPub.samples.api.
outputType: Pour les types %Status, la valeur par défaut est **header**. Pour les autres types, la valeur par défaut est **application/json**. Si vous souhaitez modifier le type de sortie, vous pouvez utiliser ce paramètre. Exemple d'utilisation : Retourner un jeton ("text/plain"). Voir la méthode ***loginUser*** de la classe apiPub.samples.api

Relier des schémas analysables à des types JSON dynamiques *(%Library.DynamicObject)*

Vous pouvez relier les schémas OAS 3.0 aux types dynamiques internes

L'avantage d'associer le schéma au paramètre, outre le fait d'informer l'utilisateur sur une spécification d'objet requise, est l'analyse automatique de la demande, qui est effectuée pendant l'appel API. Si l'utilisateur de l'API, par exemple, soumet une propriété qui ne figure pas dans le schéma, ou envoie une date dans un format non valide, ou n'inclut pas une propriété obligatoire, une ou plusieurs erreurs seront renvoyées à l'utilisateur contenant des informations sur ces problèmes.

La première étape consiste à inclure le schéma souhaité dans le bloc XDATA, comme indiqué ci-dessous. Dans ce cas, le schéma appelé *User* peut être utilisé par n'importe quelle méthode. Il doit suivre les mêmes règles que celles utilisées dans la modélisation [OAS 3.0] (https://swagger.io/docs/specification/data-models/).

```
XData apiPub [ MimeType = application/json ]
{
    {
        "schemas": {
            "User": {
                "type": "object",
                "required": [
                    "id"
                ],
                "properties": {
                    "id": {
                        "type": "integer",
                        "format": "int64"
                    },
                    "username": {
                        "type": "string"
                    },
                    "firstName": {
                        "type": "string"
                    },
                    "lastName": {
                        "type": "string"
                    },
                    "email": {
                        "type": "string"
                    },
                    "password": {
                        "type": "string"
                    },
                    "phone": {
                        "type": "string"
                    },
                    "userStatus": {
                        "type": "integer",
                        "description": "(short) User Status"
                    }
                }
            }
        }
    }
}
```

La deuxième étape consiste à associer le nom du schéma renseigné à l'étape précédente au paramètre interne de type %Library.DynamicObject en utilisant la notation suivante :

/// @apiPub[params.*paramId.schema*="*schema name*"]

Exemple associant le paramètre *user* au schéma *User* :

```
/// @apiPub[params.user.schema="User"]
Method updateUserUsingOASSchema(username As %String, user As %Library.DynamicObject)
As %Status [ WebMethod ]
{
    code...
}
```

Exemple de soumission d'une requête avec une erreur. La propriété username2 n'existe pas dans le schéma *User*. La propriété id n'est pas non plus définie alors qu'elle est requise :

```
{
  "username2": "devecchijr",
  "firstName": "claudio",
  "lastName": "devecchi junior",
  "email": "devecchijr@gmail.com",
  "password": "string",
  "phone": "string",
  "userStatus": 0
}
```

Exemple d'une réponse avec une erreur :

```
{
  "statusCode": 0,
  "message": "ERROR #5001: <Bad Request> Path User.id is required; Invalid path: User
.username2",
  "errorCode": 5001
}
```

Voir les méthodes ***updateUserUsingOASSchema*** et ***getInventory*** de la classe apiPub.samples.api. La méthode ***getInventory*** est un exemple de schéma associé à la sortie de la méthode (réponse), elle n'est donc pas analysable.

Générer le schéma OAS 3.0 à partir d'un objet JSON

Pour faciliter la génération du schéma OAS 3.0, vous pouvez utiliser les éléments suivants ::

Définissez une variable avec un échantillon de l'objet JSON.

```
set myObject = {"prop1":"2020-10-15","prop2":true, "prop3":555.55, "prop4":["banana",
```

```
"orange","apple"]}
```

Utilisez la méthode utilitaire de la classe apiPub.core.publisher pour générer le schéma :

```
do ##class(apiPub.core.publisher).TemplateToOpenApiSchema(myObject,"objectName",.sche
ma)
```

Copiez et collez le schéma renvoyé dans le bloc XDATA :

Exemple:

```
XData apiPub [ MimeType = application/json ]
{
    {
        "schemas": {
            {
                "objectName":
                {
                    "type":"object",
                    "properties":{
                        "prop1":{
                            "type":"string",
                            "format":"date",
                            "example":"2020-10-15"
                        },
                        "prop2":{
                            "type":"boolean",
                            "example":true
                        },
                        "prop3":{
                            "type":"number",
                            "example":555.55
                        },
                        "prop4":{
                            "type":"array",
                            "items":{
                                "type":"string",
                                "example":"apple"
                            }
                        }
                    }
                }
            }
        }
    }
}
```

Activer la surveillance *(facultatif)*

1 - Ajoutez et activez les composants suivants dans votre *Production* (*IRIS Interopérabilité*)

Composant	Type
apiPub.tracer.bm	Service (BS)
apiPub.tracer.bs	Service (BS)
apiPub.tracer.bo	Opération (BO)

2 - Activez la surveillance de la classe décrite dans [l'étape 2] (https://github.com/devecchijr/apiPub#passo-2)

Le paramètre ***Traceable*** doit être activé.

```
Parameter Traceable As %Boolean = 1;

Parameter TracerBSName = "apiPub.tracer.bs";

Parameter APIDomain = "samples";
```

Le paramètre ***APIDomain*** est utilisé pour regrouper les API à surveiller.

3 - Importez les tableaux de bord

```
zn "IRISAPP"
Set sc = ##class(%DeepSee.UserLibrary.Utils).%ProcessContainer("apiPub.tracer.dashboards",1)
```

Vous pouvez également créer d'autres tableaux de bord, basés sur le cube ***apiPub Monitor***.

Utilisez cet outil conjointement avec Intersystems API Manager

Acheminez vos API générées et bénéficiez de l'[Intersystems API Manager] (https://docs.intersystems.com/irislatest/csp/docbook/DocBook.UI.Page.cls...)

Compatibilité

ApiPub est compatible avec Intersystems IRIS ou Intersystems IRIS pour la santé, à partir de la version 2018.1.

Dépôt

Github: apiPub

#API #Framework #Outils #REST API #InterSystems IRIS #InterSystems IRIS for Health
Voir l'application sur InterSystems Open Exchange

URL de la source:https://fr.community.intersystems.com/post/comment-publier-rapidement-des-restful-api-dans-oas-30-%C3%A0-laide-de-loutil-iris-apipub