Artigo
Claudio Devecchi · Out. 13, 2020 15min de leitura

Open Exchange

# Como publicar rapidamente APIs Restful em OAS 3.0 utilizando a ferramenta IRIS ApiPub

## Introdução

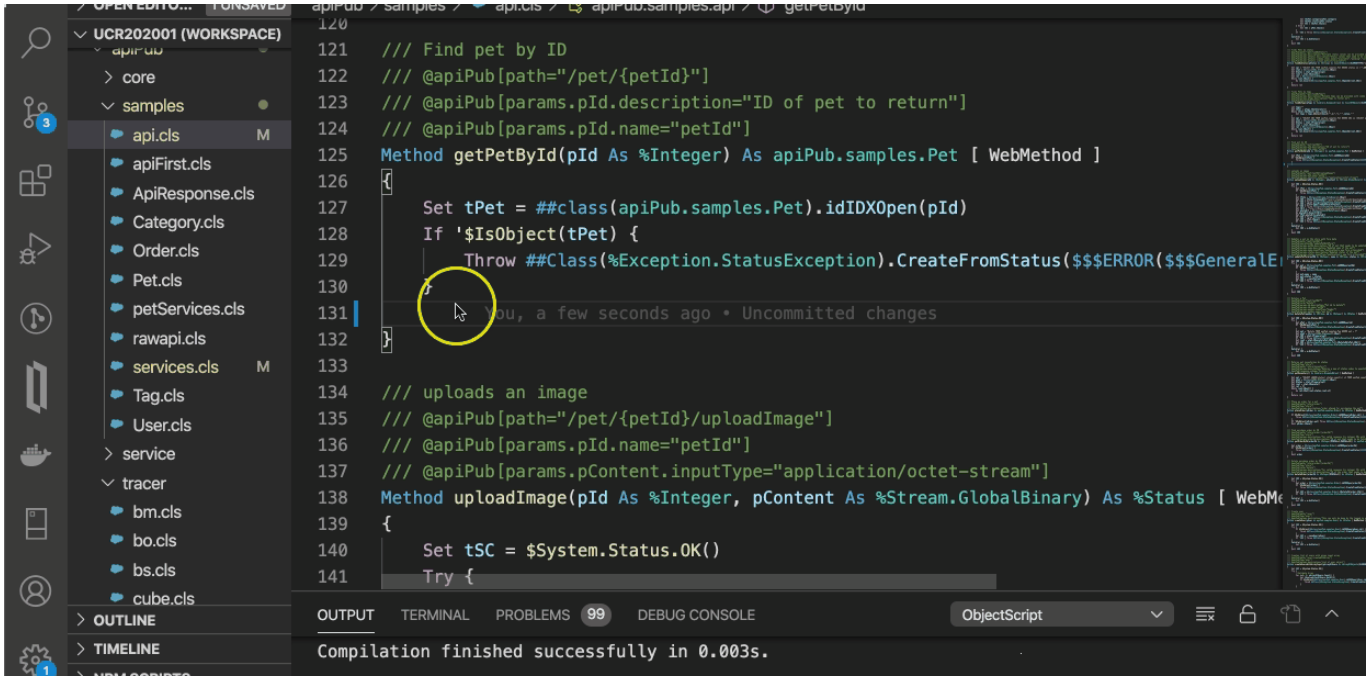
Estamos na era da **economia multi-plataforma**, e as API's são a *"liga"* deste cenário digital. Sendo tão importantes, elas são encaradas por desenvolvedores como um **serviço** ou **produto** a ser consumido. Assim sendo, a **experiência** na sua utilização é um fator crucial de **sucesso**.

Visando melhorar esta experiência, padrões de especificação, como o OpenAPI Specification (OAS) estão cada vez mais sendo adotados no desenvolvimento de API's RESTFul.

## O que é o IRIS ApiPub?

IRIS ApiPub é um projeto Open Source, que tem como principal objetivo **publicar** automaticamente **API's RESTful** criadas com a tecnologia Intersystems IRIS, da forma mais simples e rápida possível, utilizando o padrão Open API Specification (OAS) versão 3.0.

Ele permite que o usuário foque principalmente na **implementação** e nas **regras de negócio** das API ' s (Web Methods), abstraindo e automatizando os demais aspectos relacionados a **documentação, exposição, execução e monitoramento** dos serviços.




Este projeto também inclui uma implementação de exemplo completa (**apiPub.samples.api**) do *sample* Swagger Petstore, utilizado como *sample* oficial do swagger.

## Faça um teste com os teus serviços SOAP existentes

Se você já possui serviços SOAP publicados, você pode testar a sua publicação com Rest/JSON com OAS 3.0.

Ao publicar métodos com tipos complexos é necessário que a classe do objeto seja uma subclasse de %XML.Adaptor. Desta maneira serviços SOAP já construídos se tornam automaticamente compatíveis.

## Monitore as tuas API's com o IRIS Analytics

Habilite o monitoramento das API's para **administrar** e **rastrear** todas as *chamadas Rest*. Monte também os seus próprios indicadores.

## Instalação

1. Faça um *Clone/git pull* do repositório no diretório local.

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

2. Abra o terminal neste diretorio e execute o seguinte comando:

```
$ docker-compose build
```

3. Execute o container IRIS com o projeto:

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

## Testando a Aplicação

Abra a URL do swagger http://localhost:52773/swagger-ui/index.html

Tente executar alguma API do Petstore, como fazer o *post* de um novo *pet*.

Veja o dashboard do apiPub Monitor. Tente fazer um drill down no domínio petStore para explorar e analisar as mensagens.

Mude ou crie metódos na classe apiPub.samples.api e volte a consultar a documentação gerada.
Repare que todas as mudanças são automaticamentes refletidas na documentação OAS ou nos schemas.

## Publique sua API no padrão OAS 3.0 em apenas 3 passos:

## Passo 1

Defina a classe de implementação das tuas API ' s **rotule** os métodos com o atributo [WebMethod]

*Caso você já possua alguma implementação com WebServices esse passo não é necessário.*

## Passo 2

Crie uma **subclasse** de apiPub.core.service e aponte a propriedade DispatchClass para a sua classe de Implementação criada anteriormente. Informe também o path de documentação OAS 3.0. Se desejar, aponte para a classe apiPub.samples.api (PetStore).

## Passo 3

Crie uma Aplicação Web e aponte a classe de Dispatch para a classe de serviço criada anteriomente.

## Utilize o Swagger

Com o iris-web-swagger-ui é possível expor a especificação do teu serviço. Basta apontar para o path de documentação e ... **VOILÁ!!**

## Defina o cabeçalho da especificação OAS

Há duas maneiras de definir o cabeçalho OAS 3.0:

A primeira é através da criação de um bloco JSON XDATA nomeado como *apiPub* na classe de implementação. Este método permite que se tenha mais de uma Tag e a modelagem é compatível com o padrão OAS 3.0. As propriedades permitidas para customização são *info, tags* e *servers*.

```
XData apiPub [ MimeType = application/json ]
{
    {
        "info" : {
            "description" : "This is a sample Petstore server.  You can find\nout mor
e about Swagger at\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" : "Everything about your Pets",
            "externalDocs" : {
            "description" : "Find out more",
            "url" : "http://swagger.io"
            }
        }, {
            "name" : "store",
            "description" : "Access to Petstore orders"
        }, {
            "name" : "user",
            "description" : "Operations about user",
            "externalDocs" : {
            "description" : "Find out more about our store",
            "url" : "http://swagger.io"
```

```
            }
        } ]
    }
}
```

A segunda maneira é através da definição de parâmetros na classe de implementação, assim como no exemplo a seguir:

```
Parameter SERVICENAME = "My Service";

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

Parameter TITLE As %String = "REST to SOAP APIs";

Parameter DESCRIPTION As %String = "APIs to Proxy SOAP Web Services 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, all rights re
served.";

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 = "Legacy Services";

Parameter TAGDOCSDESCRIPTION As %String = "Find out more";

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

## Customize as tuas API's

É possível customizar vários aspectos das API's, como ***tags, paths e verbos***. Para tal, é necessária a utilização de uma notação específica, definida no comentário do método a ser customizado.

Sintaxe:

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

}
```

Todas as customizações dadas como exemplo nesta documentação estão disponíveis na classe apiPub.samples.api.

## Customizando os Verbos

Quando não há nenhum tipo complexo como parâmetro de entrada, apiPub atribui automaticamente o verbo como *Get*. Caso contrário é atribuído o verbo *Post*.

Caso se queira customizar o método adiciona-se a seguinte linha nos comentários do método.

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

Onde *verb* pode ser **get, post, put, delete ou patch**.

Exemplo:

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

## Customizando os Caminhos (Paths)

Esta ferramenta atribui automaticamente os *paths* ou o roteamento para os *Web Methods*. Ele utiliza como padrão o nome do método como *path*.

Caso se queira customizar o **path** adiciona-se a seguinte linha nos comentários do método.

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

Onde *path* pode ser qualquer valor precedido com barra, desde que não conflita com outro *path* na mesma classe de implementação.

Exemplo:

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

Outro uso bastante comum do path é definir um ou mais parâmetros no próprio path. Para tal, é necessário que o nome do parâmetro definido no método esteja entre chaves.

Exemplo:

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

Quando o nome do parâmetro interno difere do nome do parâmetro exposto, pode-se equalizar o nome conforme exemplo a seguir:

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

No exemplo acima, o parâmetro interno *pId* é exposto como *petId*.

## Customizando as Tags

É possível definir a **tag**(agrupamento) do método quando há mais que uma tag definida no cabeçalho.

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

Exemplo:

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

## Customizando o *Status Code* de Sucesso

Caso se queira alterar o *Status Code* sucesso do método, que é por padrão ***200***, utiliza-se a seguinte notação.

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

Exemplo:

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

## Customizando *Status Codes* de Exceção

Esta ferramenta assume como padrão o ***Status Code 500*** para quaisquer exceções. Caso se queira adicionar novos códigos para exceção na documentação, utiliza-se a seguinte notação.

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

Onde a propriedade *statusCodes* é um array de objetos com código e descrição.

Exemplo:

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

Ao disparar a exceção, Inclua o *Status Code* na descrição da exceção entre os sinais de "<" e ">".

Exemplo:

```
Throw ##Class(%Exception.StatusException).CreateFromStatus($$$ERROR($$$GeneralError, "<400> Invalid ID supplied"))}
```

Veja o método ***getPetById*** da classe apiPub.samples.api

## Marcando a API como Descontinuada

Para que a API seja exposta como ***deprecated***, utiliza-se a seguinte notação:

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

## Customizando o *operationId*

Segundo a especificação OAS, ***operationId*** é uma string única usada para identificar uma API ou operação. Nesta ferramenta ela é utilizada para a mesma finalidade no monitoramento e rastreamento das operações.

Por padrão, ela recebe o mesmo nome do método da classe de implementação.

Caso se queira alterá-la utiliza-se a seguinte notação

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

## Alterando o charset do método

O charset padrão da geralmente é definido através do parâmetro CHARSET na classe de serviço, descrita no Passo 2. Caso se queira customizar o charset de um método, deve se utilizar a seguinte notação:

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

Exemplo:

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

## Customizando nomes e outras funcionalidades dos parâmetros

Pode-se customizar vários aspectos de cada parâmetro de entrada e saída dos métodos, como por exemplo os nomes e as descrições que serão expostas para cada parâmetro.

Para se customizar um parametro específico utiliza-se a seguinte notação

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

ou para respostas:

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

Exemplo:

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

Neste caso, está sendo atribuido o nome *petId* e a descrição *ID of pet to return* para o parâmetro definido como *pId*

Quando a customização não é específica para um determinado parâmetro, utiliza-se a seguinte notação

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

No exemplo abaixo, a descrição *This can only be done by the logged in user* é atribuída para todo o *request*, não apenas para um parâmetro:

```
/// @apiPub[params.description="This can only be done by the logged in user."]
```

## Outras Propriedades que podem ser customizadas para parâmetros específicos

Utilize a seguinte notação para parâmetros de entrada ou saída:

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

Para respostas:

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

Propriedade
***required***: "true" se o parâmetro for requerido. Todos os parâmetros do tipo **path** já são automaticamente requeridos
***schema.items.enum***: exposição de Enumeradores para tipos %String ou %Library.DynamicArray. Veja o método ***findByStatus*** da classe apiPub.samples.api
***schema.default***: Aponta para um valor default para enumeradores.
***inputType***: Por padrão é **query parameter** para os tipos simples e **application/json** para os tipos complexo (body). Caso se queira alterar o tipo de input, pode se utilizar este parâmetro. Exemplo de uso: Upload de uma imagem, que normalmente não é do tipo JSON. Veja método ***uploadImage*** da classe apiPub.samples.api.
***outputType***: Por padrão é **header** para os tipos %Status e **application/json** para o restante. Caso se queira alterar o tipo de output, pode se utilizar este parâmetro. Exemplo de uso: Retorno de um token ("text/plain"). Veja método ***loginUser*** da classe apiPub.samples.api

## Relacione Schemas Parseáveis a tipos JSON Dinâmicos *(%Library.DynamicObject)*

É possível relacionar schemas OAS 3.0 a tipos dinâmicos internos.

A vantagem de se relacionar o schema com o parâmetro, além de informar ao usuário a especificação do objeto requerido, é o ***parsing automático*** do request é realizado na chamada da API. Se o usuário da API por exemplo enviar uma propriedade que não está no schema, enviar uma data em um formato inválido ou não enviar uma propriedade obrigatória, um ou mais erros serão retornados ao usuário informando as irregularidades.

O primeiro passo é incluir o schema desejado no bloco XDATA conforme exemplo abaixo. Neste caso o schema chamado *User* pode ser utilizado por qualquer método. Ele deve seguir as mesmas regras da modelagem OAS 3.0
.

```
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"
                    }
                }
            }
        }
    }
}
```

O segundo passo é relacionar o nome do schema informado no passo anterior ao parâmetro interno do tipo %Library.DynamicObject usando a seguinte notação:

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

Exemplo associando o parâmetro *user* ao schema *User*:

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

Exemplo de request com erro a ser submetido. A propriedade username2 não existe no schema *User*. A propriedade id também não foi especificada e é requerida:

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

Exemplo de erro retornado:

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

Veja métodos ***updateUserUsingOASSchema*** e ***getInventory*** da classe apiPub.samples.api. O método ***getInventory*** é um exemplo de schema associado à saída do método (response), portanto não é parseável.

## Gere o schema OAS 3.0 com base em um objeto JSON

Para auxiliar na geração do schema OAS 3.0, você pode usar o seguinte recurso:

**Defina** uma variável com uma amostra do objeto JSON.

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

**Utilize o método utilitário** da classe apiPub.core.publisher para gerar o schema:

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

```
ma)
```

**Copie e cole** o schema retornado no bloco XDATA:

Exemplo:

```
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"
                            }
                        }
                    }
                }
            }
        }
    }
}
```

## Habilite o Monitoramento *(Opcional)*

1 - Adicione e ative os seguintes componentes na tua *Production* (*IRIS Interoperability*)

| Component | Type |
|---|---|
| apiPub.tracer.bm | Service (BS) |
| apiPub.tracer.bs | Service (BS) |
| apiPub.tracer.bo | Operation (BO) |

2 - Ative o monitoramento na classe descrita no Passo 2

O parâmetro ***Traceable*** deve estar ativado.

```
Parameter Traceable As %Boolean = 1;
```

```
Parameter TracerBSName = "apiPub.tracer.bs";

Parameter APIDomain = "samples";
```

O parâmetro ***APIDomain*** é utilizado para agrupar as API's no monitoramento.

3 - Importe os dashboards

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

Outros dashboards também podem ser criados com base no cubo ***apiPub Monitor***.

## Utilize esta ferramenta em conjunto com o Intersystems API Manager

Roteie as suas API's geradas e obtenha diversas vantagens com o Intersystems API Manager

## Compatibilidade

ApiPub é compatível com o produto Intersystems IRIS ou Intersystems IRIS for Health a partir da versão 2018.1.

## Repositório

***Github***: apiPub

#API #Ferramentas #Framework #REST API #InterSystems IRIS #InterSystems IRIS for Health
Confira o aplicativo relacionado no InterSystems Open Exchange

---

**URL de origem:**https://pt.community.intersystems.com/post/como-publicar-rapidamente-apis-restful-em-oas-30-utilizando-ferramenta-iris-apipub