REST API Documentation Using OpenAPI Version 3.0

Martyn Kemp
Senior Consultant
Consultingwerk Software Services Ltd.

- Independent IT consulting organization
- Focusing on OpenEdge and related technology
- Located in Cologne, Germany, subsidiaries in UK and Romania
- Customers in Europe, North America, Australia and South Africa
- Vendor of developer tools and consulting services
- Specialized in GUI for .NET, Angular, OO, Software Architecture, Application Integration
- Experts in OpenEdge Application Modernization
Martyn Kemp

- Senior Consultant
- 20 plus years of development experience in various roles and using various technologies
- Initially started using Progress just before the release of version 7.
- First started using the .Net Framework with the introduction of ProDatasets (circa 2002). During those times and up to the current time, Martyn has developed using OpenEdge GUI for .Net, C#, VB .Net and more recently, OE Mobile, OE REST, Spring IO, PASOE, Angularjs 1.x & Angular 2 and Swagger. Martyn currently works as Senior Consultant for Consultingwerk.
Agenda

- Introduction
- Resources
- Definition Structure
- Online Editor
- Consultingwerk OpenAPI
- Using Consultingwerk Swagger UI with JSDO Generic Services
- Using Consultingwerk Swagger UI with RESTful Services
- OpenEdge GENOAS
- Upgrade Swagger to OpenAPI
Introduction

- What is OpenAPI?
  - The OpenAPI Specification (formerly Swagger) is an API description format for REST APIs.
  - Allows you to describe your entire API
  - Enables development across the entire API lifecycle, from design and documentation, to test and deployment
  - A powerful definition format for describing and creating RESTful API’s, which are easy to understand, readable and language agnostic
Introduction

What is Swagger?

- Swagger is a set of open-source tools built around the OpenAPI Specification
- Enables you to design, build, document and consume your REST API’s
  - The main Swagger Tools include:
    - Swagger Editor – browser-based editor where you can write OpenAPI specs.
    - Swagger UI – renders OpenAPI specs as interactive API documentation.
    - Swagger Codegen – generates server stubs and client libraries from an OpenAPI spec.

- API specifications can be written in JSON or YAML (Yet Another Markup Language). The format is easy to learn and readable to both humans and machines.
Why Use OpenAPI

- **Faster, Standardized API Design**
  - Design APIs in a powerful and intuitive editor that is built for speed and efficiency, without any loss in design consistency

- **Centralized, Secure API Collaboration**
  - Seamlessly work across multiple teams on your API development with controlled, centralized access and an optimized collaborative workflow

- **Hosted, Interactive API Documentation**
  - Autogenerate interactive API Documentation straight from the contract and securely host it, making them easy to use and adopt by internal and external users
Why Use OpenAPI

- **Design**
  - Commonly achieved through the Swagger Editor
  - [http://editor2.swagger.io](http://editor2.swagger.io)
  - Using the on line editor, you can edit existing API's or design new API's, which visually renders your Swagger Definition with real time feedback and error handling

- **Build**
  - Generate code from your API’s using the Swagger Codegen Tool (not for OE)

- **Document**
  - Using the Swagger UI, you can visualise and interact with your Swagger Definitions
Agenda

- Introduction
- **Resources**
- Definition Structure
- Online Editor
- Consultingwerk OpenAPI
- Using Consultingwerk Swagger UI with JSDO Generic Services
- Using Consultingwerk Swagger UI with RESTful Services
- OpenEdge GENOAS
- Upgrade Swagger to OpenAPI
Swagger – Resources

- The OpenAPI Specification can be found on GitHub:
  - [OpenAPI 3.0 Specification](https://github.com/OASpecification/openapi-3.0)

- Swagger UI
  - Swagger Editor [https://swagger.io/swagger-editor/](https://swagger.io/swagger-editor/)
    - It’s clean, efficient, and armed with a number of features to help you design and document your RESTful interfaces, straight out of the.

- Swagger Codegen [https://swagger.io/swagger-codegen/](https://swagger.io/swagger-codegen/)
  - Can simplify your build process by generating server stubs and client SDKs from your OpenAPI specification. Can also generate client libraries for your API in over 40 languages
Swagger – Resources

- **Swagger UI** [https://swagger.io/swagger-ui/](https://swagger.io/swagger-ui/)
  - Allows your development team or your end consumers — to visualize and interact with the API’s resources without having any of the implementation logic in place. Also lets your users try out the API calls directly in the browser.

- **Swagger Inspector** [https://swagger.io/swagger-inspector/](https://swagger.io/swagger-inspector/)
  - An inspection tool for easily calling and validating REST, GraphQL and SOAP based web services to ensure they function correctly.
Swagger – Resources

- **Swagger Hub** - [https://swaggerhub.com](https://swaggerhub.com)
  - Platform for Designing and Documenting API’s
  - Fastest way for teams to collaborate on their API Development

- **Swagger Blog** - [https://swaggerhub.com/blog/](https://swaggerhub.com/blog/)
  - Area for keeping up to date on activities and events within the Swagger Community

- **Swagger Docs**
  - [https://swagger.io/docs/](https://swagger.io/docs/)

- **Swagger Pet Store Sample**
  - [https://petstore.swagger.io/](https://petstore.swagger.io/)
Resources

- **Swagger OpenAPI Converter**
  - [https://mermade.org.uk/openapi-converter](https://mermade.org.uk/openapi-converter)
  - On Line Engine Tool for converting Swagger 2.0 to OpenAPI 3.0.0
  - Free …..
  - The converter tool shall be demonstrated later in the comparison section
Agenda

- Introduction
- Resources
- **Definition Structure**
- Online Editor
- Consultingwerk OpenAPI
- Using Consultingwerk Swagger UI with JSDO Generic Services
- Using Consultingwerk Swagger UI with RESTful Services
- OpenEdge GENOAS
- Upgrade Swagger to OpenAPI
## Basic Structure

<table>
<thead>
<tr>
<th>Swagger 2.0</th>
<th>OpenAPI 3.0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Metadata</td>
<td>Metadata</td>
</tr>
<tr>
<td>Base URL</td>
<td>Servers</td>
</tr>
<tr>
<td>Consumes, Produces</td>
<td>-</td>
</tr>
<tr>
<td>Paths</td>
<td>Paths</td>
</tr>
<tr>
<td>Operations</td>
<td>Operations</td>
</tr>
<tr>
<td>Parameters &amp; Parameter Types</td>
<td>Parameters &amp; Parameter Types</td>
</tr>
<tr>
<td>Responses &amp; Response Media Types</td>
<td>Responses &amp; Response Media Types</td>
</tr>
<tr>
<td>Input, Output Models (Definitions)</td>
<td>Input, Output Models (Definitions)</td>
</tr>
</tbody>
</table>
Structure

Swagger 2.0

```json
swagger: '2.0'
info:
  version: 1.0.0
  title: Swagger Consultingwerk
  description: Restful Services Entities
  termsOfService: 'http://consultingwerk/terms/

schemes: [
  "http"
]
host: "localhost:8820"
basePath: "/web/Entities"
paths:
  /Customers:
    get:
      tags:
        - CustomerBusinessEntity
      summary: Fetch Data
      description: Fetch Customers
      operationId: FetchCustomers
      parameters:
        - in: query

OpenAPI 3

```json
openapi: 3.0.0
info:
  version: 1.0.0
  title: Swagger Consultingwerk
  description: Restful Services Entities
  termsOfService: 'http://consultingwerk/terms/

schemes:
  - http

servers:
  - url: 'http://localhost:8820/web/Entities'
    description: Consultingwerk API

paths:
  /Customers:
    get:
      tags:
        - CustomerBusinessEntity
      summary: Fetch Data
      description: Fetch Customers
      operationId: FetchCustomers
      parameters:
        - in: query
```
Metadata

- Details:
  - https://swagger.io/docs/specification/2-0/basic-structure/

- Every Swagger specification starts with the Swagger version. A Swagger version defines the overall structure of an API specification — what you can document and how you document it.

  - `swagger: "2.0"`

- Then you need to specify the “info” metadata tag. This is an object that should contain “title” and “version”

  - `info:
    title: Customer API
    description: Customer API Definition.
    version: 1.0.0`

- Details:
  - https://swagger.io/docs/specification/basic-structure/

- Every OpenAPI specification starts with the ‘openapi’ keyword and version. The version defines the overall structure of an API specification — what you can document and how you document it. Since version 3, semantic versioning is now being used.

  - `openapi: 3.0.0`

- Then you need to specify the “info” metadata tag. This is an object that should contain “title” and “version”

  - “Same as in Version2”
Swagger – Base URL

- **Base URL (in OpenAPI – this is now deprecated)**
  - The base URL for all API calls is defined using schemes, host and basePath:
    ```json
    "schemes": [
    "http"
    ],
    "host": "localhost:8820",
    "basePath": "/web/Entities",
    ```
  - **Note:** if “host” is not specified, then it is assumed to be the same “host” value that is currently serving the API Documentation

- **basePath:**
  - “basePath” is basically the URL prefix for ALL API Paths relative to the “host” (and must begin with “/”)
  - An Example of the concatenated path would be “http://machine:port/web/resource”
  - **Note:** If both “host” and “scheme” are omitted, then the values are derived from machine serving the API Document
OpenAPI – Servers

- Base URL

- The ‘servers’ section specifies the API server and base URL. You can define one or several servers, such as production and sandbox.
  - All API paths are relative to the server URL.

  Servers:
  - url: http://localhost:8820/web
  - url: https://localhost:8821/web

- Note: Unlike in Swagger 2.0, the Servers MUST be specified
Consumes & Produces

- **Defunct in OpenAPI 3.0**
  - The “consumes” and “produces” sections define the MIME (Multipurpose Internet Mail Extensions) types supported by the API
    
    ```json
    "consumes": ["application/json"],
    "produces": ["application/json"],
    ```
  
  - The value of consumes and produces is an array of MIME types
  - Global MIME types can be defined on the root level of an API specification and are inherited by all API operations. However, these can be overridden at operation level
Paths

- In the OpenAPI definition API, paths are resources that your API exposes.

- A single path can support multiple operations, however, Swagger defines a unique operation as a combination of a path and an HTTP method. This means that two GET or two POST methods for the same path are not allowed.

- This means that you have to define alternative paths for addition operations, e.g:

```
"/Customers": {
  "get": {}, "put": {}, "post": {}, "delete": {}
}
"/Customers/Count": {
}
"/Customers/Submit": {
}
```

- The full request URL is constructed as:
  - Swagger 2.0: scheme://host/basePath/path
  - OpenAPI 3.0: <server-url>/path
Operations

- For each path, you define operations (HTTP methods) that can be used to access that path. Swagger & OpenAPI supports get, post, put, patch, delete, head, options and trace.
- Addition resources, such as "Submit", "Count", "Invoke" (Invokable Methods) are defined as additional paths

```
"/Customers/Count": {
  "get": {}
}
"/Customers/{CustNum}/PutCustomerOnHold": {
  "get": {}, “post”: {},
}
```
Parameters

- API operation parameters are defined under the parameters section in the operation definition. Each parameter has name, value type (for primitive value parameters) or schema (for request body), and optional description.

```json
parameters:
  - in: body
    name: eCustomer
    required: true
    description: Records to Create
    schema:
      $ref: '#/definitions/dsCustomer/properties/eCustomer'
```

- **Note**: *Parameters is an Array of one or more objects*
Parameter Types

- Swagger distinguishes between the following parameter types based on the parameter location. The location is determined by the parameter’s “in” key, for example:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Path</td>
<td>/customer/{custnum}</td>
</tr>
<tr>
<td>Query</td>
<td>/customers?filter={filter}</td>
</tr>
<tr>
<td>Header</td>
<td>X-Header: some value</td>
</tr>
<tr>
<td>Cookie</td>
<td>Cookie: jSessionID=ABC12345XYZjs</td>
</tr>
</tbody>
</table>

- Required and Optional Parameters
  
  By default, all request parameters as optional. You can add required: true to mark a parameter as required. Note that path parameters must have required: true, because they are always required

  “parameters”: [{“...,” “required”: true, ..... }]
## Parameter Types

- **Required and Optional Parameters**

  By default, all request parameters as optional. You can add `required: true` to mark a parameter as required. Note that path parameters must have `required: true`, because they are always required.

<table>
<thead>
<tr>
<th>Swagger</th>
<th>OpenAPI</th>
</tr>
</thead>
<tbody>
<tr>
<td>get:</td>
<td>get:</td>
</tr>
<tr>
<td>summary: Gets a user by Custnum.</td>
<td>summary: Gets a user by Custnum.</td>
</tr>
<tr>
<td>parameters:</td>
<td>parameters:</td>
</tr>
<tr>
<td>- in: path</td>
<td>- in: path</td>
</tr>
<tr>
<td>name: userId</td>
<td>name: userId</td>
</tr>
<tr>
<td><strong>required: true</strong></td>
<td><strong>required: true</strong></td>
</tr>
<tr>
<td><strong>type: integer</strong></td>
<td><strong>type: integer</strong></td>
</tr>
<tr>
<td>description: Value of the user to retrieve</td>
<td>description: Value of the user to retrieve</td>
</tr>
</tbody>
</table>
**Responses**

- An API specification needs to specify the responses for all API operations. Each operation must have at least one response defined, usually a successful response. A response is defined by its HTTP status code and the data returned in the response body and/or headers.

<table>
<thead>
<tr>
<th>Swagger</th>
<th>OpenAPI</th>
</tr>
</thead>
<tbody>
<tr>
<td>responses: '200':</td>
<td>responses: '200':</td>
</tr>
<tr>
<td>description: successful operation</td>
<td>description: successful operation</td>
</tr>
<tr>
<td>content:</td>
<td>content:</td>
</tr>
<tr>
<td>text/plain:</td>
<td>text/plain:</td>
</tr>
<tr>
<td>schema:</td>
<td>schema:</td>
</tr>
<tr>
<td>type: string</td>
<td>type: string</td>
</tr>
</tbody>
</table>
Response Media Types

- An API specification needs to specify the responses for all API operations. Each operation must have at least one response defined, usually a successful response. A response is defined by its HTTP status code and the data returned in the response body and/or headers.
- An API can respond with various media types. JSON is the most common format, but certainly not the only one.

**Swagger & OpenAPI**

```json
responses:
  '200':
    description: A list of users
    content:
      application/json:
        schema:
          $ref: '#/definitions/dsCustomer/properties/eCustomer'
      text/plain:
        schema:
          type: string
```
Response Body

- The schema keyword is used to describe the response body
- A schema can define:
  - An “object” or “array” – this format is usually represented with JSON and XML API’s
  - A primitive such as a number or string – this is typically used for plain text responses
- A schema can be defined in two possible ways
  - inline
  - $sref (preferred choice as this prevents duplicate and groups definitions together)
inline Response Body

responses:
'200':
description: A list of users
content:
application/json:
application/json:
schema
type: array
items:
  additionalProperties: true
properties:
id:
  type: string
url:
  type: string
Country:
  type: string
x-ablType: CHARACTER
default: USA
title: Country
$ref Response Body

- $sref

'200':
  description: A list of users
  content:
    application/json:
      schema:
        $ref: '#/definitions/dsCustomer/properties/eCustomer'
Input & Output Models

- The global definitions section lets you define common data structures used in your API. They can be referenced via $ref whenever a schema is required:
  - request body, response body and even parameters (body).

- For example, let's suppose we have the following object:

```json
{
    "CustNum": 13,
    "Name": "Freddy Krugger",
    "Comments": "Friendly Psychopath …"
}
```
## Input & Output Models

- The above JSON object can then be represented as:

<table>
<thead>
<tr>
<th>Swagger</th>
<th>OpenAPI</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>components:</td>
</tr>
<tr>
<td></td>
<td>schemas:</td>
</tr>
<tr>
<td></td>
<td>dsCustomer:</td>
</tr>
<tr>
<td></td>
<td>type: object</td>
</tr>
<tr>
<td></td>
<td>additionalProperties: false</td>
</tr>
<tr>
<td></td>
<td>properties:</td>
</tr>
<tr>
<td></td>
<td>eCustomer:</td>
</tr>
<tr>
<td></td>
<td>type: array</td>
</tr>
<tr>
<td></td>
<td>items:</td>
</tr>
<tr>
<td></td>
<td>additionalProperties: false</td>
</tr>
<tr>
<td></td>
<td>properties:</td>
</tr>
<tr>
<td></td>
<td>CustNum:</td>
</tr>
<tr>
<td></td>
<td>type: integer</td>
</tr>
<tr>
<td></td>
<td>default: 0</td>
</tr>
<tr>
<td></td>
<td>Name:</td>
</tr>
<tr>
<td></td>
<td>type: string</td>
</tr>
<tr>
<td></td>
<td>default: &quot;&quot;</td>
</tr>
<tr>
<td></td>
<td>Comments:</td>
</tr>
<tr>
<td></td>
<td>type: string</td>
</tr>
</tbody>
</table>

```json
The above JSON object can then be represented as:

```

```json
```
Grouping Operations

- Grouping Operations With Tags
  - The OpenAPI specification allows API operations to be grouped together by the use of "tags". For example, Swagger UI uses tags to sort and group the displayed operations
  - “Tags” can be defined globally and referenced within the relevant “Operation”
    - Note: The “tags” attributes should match
  - “Tags” can be defined within each “Operation” (without global “tags”)
Grouping “tags”

- Global “tags”

```plaintext
tag:
description: 'Example Business Entity for Customer, read only access to Salesrep'
description: 'Example Business Entity for Orders and OrderLines'
description: 'Example Business Entity for Items'
```
Operation “tags”

- Operation “tags”

/Customers:
get:
tags:
post:
tags:

/Orders:
get:
tags:
post:
tags:
“tags”

- Regardless of which method you invoke, the final display shall always be represented as seen below

- The benefit of using “Global ‘tags’” is that duplication is reduced (e.g. Doc Ref, Descriptions)
Agenda

- Introduction
- Resources
- Definition Structure
- **Online Editor**
- Consultingwerk OpenAPI
- Using Consultingwerk Swagger UI with JSDO Generic Services
- Using Consultingwerk Swagger UI with RESTful Services
- OpenEdge GENOAS
- Upgrade Swagger to OpenAPI
Swagger – Demo

- Demonstrate the Swagger Editor
Manual or Dynamic Design

▪ So why not manual design?
  ▪ Difficult to maintain (bad memories....)
  ▪ Version Control
  ▪ Etc ....

▪ Dynamic
  ▪ No need to worry about changes
  ▪ Bug Fixes and Enhancements are centralised (meaning ALL future OpenAPI definitions automatically see changes)

▪ We do not hand crank our OpenAPI definitions. Our OpenAPI definitions are generated dynamically through OpenEdge code
Agenda

- Introduction
- Resources
- Definition Structure
- Online Editor
- **Consultingwerk OpenAPI**
  - Using Consultingwerk Swagger UI with JSDO Generic Services
  - Using Consultingwerk Swagger UI with RESTful Services
  - OpenEdge GENOAS
  - Upgrade Swagger to OpenAPI
Consultingwerk Swagger

- Consultingwerk now provides OpenAPI (Swagger) Documentation as part of the SmartComponent Library framework
- Consultingwerk provides two Variations of the OpenAPI (Swagger) Documentation
  - Swagger REST API Documentation for JSDO Generic Service
    - [https://documentation.consultingwerkcloud.com/display/SCL/Swagger+REST+API+Documentation+for+JSDO+Generic+Service](https://documentation.consultingwerkcloud.com/display/SCL/Swagger+REST+API+Documentation+for+JSDO+Generic+Service)
  - Swagger REST API Documentation for RESTful Services
    - Uses the HATEOAS Driven REST API’s standard
      - HATEOAS (Hypermedia as the Engine of Application State)
      - Reference: [https://restfulapi.net/hateoas/](https://restfulapi.net/hateoas/)
    - [https://documentation.consultingwerkcloud.com/display/SCL/Swagger+REST+API+Documentation+for+RESTful+Services](https://documentation.consultingwerkcloud.com/display/SCL/Swagger+REST+API+Documentation+for+RESTful+Services)
Agenda

- Introduction
- Resources
- Definition Structure
- Online Editor
- Consultingwerk OpenAPI
- Using Consultingwerk Swagger UI with JSDO Generic Services
- Using Consultingwerk Swagger UI with RESTful Services
- OpenEdge GENOAS
- Upgrade Swagger to OpenAPI
Consultingwerk Swagger REST API for JSDO Generic Service

- REST API Documentation for JSDO Generic Service
  - The Consultingwerk Swagger Generator produces REST API Swagger Documentation for SmartComponent Library Business Entities, allowing the consumer to understand and interact with the remote service through the Generic Service Interface for the JSDO or Pacific WebSpeed WebHandler based JSDO Generic Service.
  - The Consultingwerk Swagger Generator is accessible dynamically through a Web Handler and can produce static Swagger definitions through a plugin for the Business Entity Designer.
  - The Consultingwerk Swagger Generator is also accessible from within the Business Entity Designer.
Consultingwerk Swagger REST API for JSDO Demo

- Demo
  - The Consultingwerk Swagger Generator from the WebHandler
Consultingwerk Swagger REST API for JSDO Generic Service

- Non JSDO works fine for ALL Operations
- Working with JSDO
  - CRUD
    - One of the main downsides encountered is the with the CRUD operations.
    - R – Fetch data works perfectly fine
    - CUD – This does not work due to the missing “useBeforeImage” attribute
      - When performing any “CUD” operation, no “before image” is created therefore, when the record hits the PAS, it’s not recognised.
Consultingwerk Swagger REST API for JSDO Generic Service

- **Filtering**
  - A typical URL Filter for accessing JSDO data would be:
  - By default, OpenAPI (Swagger) does not accept JSON strings within the URL. Therefore, in order for the above to work as a “filter” within Swagger, we need to construct a JSON Object. This in turn means that the Filter Parameter is rather restrictive as we have to specify examples and rely on correct user input.
Consultingwerk Swagger REST API for JSDO Generic Service

- Authentication Token
  - Swagger does include the facility for implementing Authentication
  - There are plenty of examples in the Swagger Forum using the Spring Framework with Swagger
  - Has not been implemented here as Authentication is dealt with before invoking a Swagger Definition.
Agenda

- Introduction
- Resources
- Definition Structure
- Online Editor
- Consultingwerk OpenAPI
- Using Consultingwerk Swagger UI with JSDO Generic Services
- Using Consultingwerk Swagger UI with RESTful Services
- OpenEdge GENOAS
- Upgrade Swagger to OpenAPI
Consultingwerk Swagger REST API for RESTful Services

- REST API Documentation for RESTful Services
  - RESTful services provide an alternative HTTP and JSON based interface for third party applications or mobile applications (the JSDO generic service is specialized to be used by JSDO based clients).

- The Consultingwerk Swagger RESTful Services Generator produces REST API Swagger Documentation for SmartComponent Library Business Entities exposed as RESTful services, allowing the consumer to understand and interact with the remote service.

- Based on the HATEOAS standard, this architectural style lets you use hypermedia links in the response contents so that the client can dynamically navigate to the appropriate resource by traversing the hypermedia links.
Consultingwerk Swagger REST API for RESTful Services – Demo

- Swagger REST API Documentation for RESTful Services
  - http://localhost:8820/web/SwaggerEntities/json
Agenda

- Introduction
- Resources
- Definition Structure
- Online Editor
- Consultingwerk OpenAPI
- Using Consultingwerk Swagger UI with JSDO Generic Services
- Using Consultingwerk Swagger UI with RESTful Services
- OpenEdge GENOAS
- Upgrade Swagger to OpenAPI
OpenEdge GENOAS

- Within the Progress Community, there exists a Javascript Library called “genoas” that can be used to generate OpenAPI documents for JSDO based catalogs.

- Generate an OpenAPI specification file from a catalog file
  - [https://community.progress.com/community_groups/mobile/m/documents/3389](https://community.progress.com/community_groups/mobile/m/documents/3389)
Agenda

- Introduction
- Resources
- Definition Structure
- Online Editor
- Consultingwerk OpenAPI
- Using Consultingwerk Swagger UI with JSDO Generic Services
- Using Consultingwerk Swagger UI with RESTful Services
- OpenEdge GENOAS
- Upgrade Swagger to OpenAPI
Demo

- Structure Comparison
  - Swagger 2.0 / OpenAPI 3.0.0
- Change the Basic Structure Definitions
  - Demonstrate the Syntax Errors
- Demo the On Line Conversion Tool
  - Run inside the editor
Questions