# SWAGGER OPENAPI SPECIFICATION 3.0 DOCUMENTATION CHEAT-SHEET:

Following are the metadata we use for documenting OASIS APIs:

* **openapi**: Every OpenAPI specification starts with the openapi keyword mentioning the version of the specification format. This version defines the overall structure of an API specification – what you can document and how you document it. This value needs to be “3.0.x”, where x is a number between 0 to 9.
* **servers**: The servers section specifies the API server and base URL. You can define one or several servers, such as SILO4, SILO5, Pre-Prod, and production. We need to describe the following properties under this heading:
  + **url:** Server name of the host, e.g. <https://tmnh7o.manheim.com>
  + **description:** Short description, e.g. SILO4 Internal End Point

In this section we specify all the host names, both internal and external.

* **info**: The info metadata contains information about the API: version, title, description, contact & license details. Following properties can be described under this heading:
  + **version**: version is an arbitrary string that specifies the version of your API. It is not the file revision or the openapi version. We will use it as our API version, e.g. 2.1, 2.2 etc
  + **title**: is the name of the API, e.g. ‘RECEIPTS API 2.5’
  + **contact**: This is the optional contact information for the functional person supporting the API. We need to describe the following properties under this heading:
    - **name:** OASIS Support
    - **email:** email address of the functional lead supporting the API
    - **url**: 'http://www.coxautoinc.com'
  + **license**: Optional license information. We can describe the following properties under this heading:
    - **name**: e.g. Cox Automotive
    - **url**: 'http://www.coxautoinc.com'
  + **description**- This section describes details of the API and can include the following type of information:
    - Release Notes
    - Purpose of the API
    - Definitions
    - Architecture
    - Versioning
    - Application Authentication
    - End User Authentication: Define for each consumer
* **externalDocs**: This section provides the details of additional external documentation information. Here, we can provide the url link to AN100 document in the sharepoint. This section can contain the following information:
  + **description:** Brief description of the external document being referenced
  + **url:** We will provide the url of any other reference document stored in the Rally/Sharepoint
* **paths:** The paths section defines individual endpoints (URI) of your API. All paths are relative to the API host server URL defined in the servers section. The full request URL is constructed as <server-url>/path, for example the path of receipts API is /receipts2.

For each path, we define the operations (HTTP methods) that can be used to access that path. OpenAPI 3.0 supports get, post, put, patch, delete, head, options, and trace. A single path can support multiple operations, for example, GET /receipts2 to get a the receipt details and POST /receipts2 to create a receipt.

The path (/receipts2) needs to be defined only one time and within that path each operation/method needs to be defined separately, e.g.

/receipts2:

parameters: Define shared parameters here that are common for all the methods. We will define header level parameters (version, application name, user id etc) here, since they are common for all the methods.

get:

…….

…….

…….

post:

…….

…….

…….

Below is the information that is defined for each method/operation (get/post/put/delete):

* + **security**: This is the type of security being used, e.g. HTTP basic authentication, OAuth2, API key etc. For most of our APIs we use HTTP basic authentication and define it as follows:
    - basicAuth: []
  + **summary**: Brief description of this API method
  + **operationId**: Enter the operation id, e.g. getReceipts, postReceipts etc.
  + **tags**: The first value should always be ‘Endpoints’ for every defined method. You can define additional tags and description if there is a need to enter additional information about API.
  + **description**: This is the brief description of the method. We define end points for each internal and external environment in this section
  + **parameters**: Methods can have parameters that are passed to the server via URL. We also define parameter data types, format, whether they are required or optional, and other details in this section. You can have shared parameters that are common to all the operations, such parameters are defined immediately after the end point URI. Any parameters specific to a method are defined here. We can have two types of parameters:
    - **in: header**: For each API that requires SOA authentication must have the below header parameters defined in this section. These header parameters are passed in httpS header of the API call request.
      * Version: Required
      * Manheim-Employee-ID: Optional
      * Manheim-Customer-Role: Optional
      * Manheim-Customer-RepNumber: Optional
      * Manheim-Customer-UserName: Optional

We can also define enums for these header parameters, wherever applicable.

Optionally, you can define a default value for a parameter by using “default: <value>”.

**Note:** All header level parameters will be defined as common parameters for all the methods.

1. **in: query**: These are the actual parameters passed in the body of the API call request. These would vary based on each the API definition, for e.g. Receipts API accepts customerNumber, receiptID, startDate, endDate etc. as input query parameters. For each these parameters we describe the following properties:
   * + - **name:** This is the name of the parameter
       - **allowEmptyValue**: boolean. This property indicates if a parameter value can be empty or not. If this is defined as true for a parameter then required should be set as false.
       - **required**: boolean. This property indicates if a parameter is required or not
       - **description**: Description of the data element
       - **schema**: Define data type details for the parameter
         * **type**: Data type of the parameter. One of the following data types that can be assigned to a parameter:

string

number

integer

boolean

array

object

* + - * + **format**: Optionally, define the format of the parameter, e.g. date or int32 etc.
      * **default**: Optionally, we can provide a default value for the parameter, e.g. for Receipts API’s latest Version parameter we can pass 2.4
      * **enum**: Optionally, define a set of restricted values that a parameter can have, e.g. outputExpansions parameter for Receipts API can accept only one of the following values- APPLICATIONS / RECEIPT DETAILS / APPLICATIONS,RECEIPT DETAILS
      * **minimum:** Optionally, specify the minimum and maximum range for values accepted by a parameter
      * **maximum:** Optionally, specify the minimum and maximum range for values accepted by a parameter
      * **example**: Optionally, you can provide a sample value of the, e.g. for customerNumber it can be “5004945”
  + **responses**: Here you define the responses returned by the API operation.  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. For each operation, you can define possible status codes, such as 200 OK or 401 Bad Request or 404 Not Found.

If an operation returns the data elements in the response, then we need to define the request body, e.g. GET method call returns the data elements for the request call. The schema keyword is used to describe the response body.

A **schema** can define:

* + - an object or an array — typically used with JSON and XML APIs,
    - a primitive data type such as a number or string – used for plain text responses

Schemas can be defined inline or referenced via [$ref](https://swagger.io/docs/specification/using-ref/).  For POST & PUT methods, you need to define the output response details for 201 status code and explain the data elements, e.g. receiptStatus, receiptID etc., returned by the API method. For other responses, we need to mention all the error details in the description field, e.g. for 400 it can have “No receipts were found for the passed input filters” and for 404 it can have “receiptID not found”. One response (400, 401 etc.) can have more than error details within one description field.

Please see receipts and/or payments API yaml file to understand how response body is defined.

* + **requestBody**: Request bodies are typically used with “create” and “update” operations (POST, PUT, PATCH). For example, when creating a resource using POST or PUT, the request body usually contains the representation of the resource to be created. In such cases, use the requestBody  to describe the body content here, i.e. request payload.

Please note that you cannot define request body with GET or DELETE operations.

Request bodies are optional by default. To mark the body as required, use property required: true.

Just like in responses & parameters properties, schemas can be defined inline or referenced via [$ref](https://swagger.io/docs/specification/using-ref/).

Please see receipts and/or payments API yaml file to understand how request body is defined.

* **components**- When you document an API, it is common to have some features which you use across several of API resources. In that case, you can create a snippet for such elements in order to use them multiple times within the yaml file, when you need it. These are kind of define once used multiple times within the API description file. We will use local reference elements and will later mature towards using remote references. To reference a definition, use the $ref keyword.

Please see receipts and/or payments API yaml file to understand how reference components are defined and used to document an API.

* **securitySchemes** – This section defines security scheme used by overall document. If any of the method of the API uses authentication then this section must be defined.

**basicAuth**:

**type**: http

**scheme**: basic

* Tip: Comments can be added in the .yaml file starting with #. There are few caveats for entering comments is the description metadata.

**References:**

* Open API Specification Documentation:
  + <https://swagger.io/docs/specification/about/>
  + <https://swagger.io/specification/>
* Cox AutoInc Portal: <http://developer.caionoci.coxautoinc.com/>
* Swagger Editor: <http://developer.caionoci.coxautoinc.com/swagger/index.html>
* Enterprise gitHub link: <https://ghe.coxautoinc.com/> . A snow ticket needs to be opened get the access to enterprise gitHub.
* CORS Chrome Extension (Cross-origin resource sharing)- <https://chrome.google.com/webstore/detail/allow-control-allow-origi/nlfbmbojpeacfghkpbjhddihlkkiljbi?hl=en>

**Swagger Editor Limitations:**

* Does not auto save the changes. You need to keep on saving changes with correct file name.
* Works best with Chrome browser.
* Install Chrome extension “Allow-Control-Allow-Origin” if you wish to use “Try it out” option.