



# DOCUMENTING APIs

**Sarah Kittrell**  
**Technical Writer**

March 2019

# WORKSHOP OBJECTIVES

- Understand APIs and their role at Finastra
- Be able to identify all of the components of an API and their purpose (what they do)
- Be familiar with the tools used to build and document APIs (Swagger, Markdown, Postman)

# AGENDA

- Introduction to APIs
- Parts of an API
- Content Objectives
- API Standards Guide (Reference Content)
- Documenting Non-Reference Content
- Content Plan Implementation



Throughout this slide deck, you will see pins dropped to indicate the slide content may require additional input from your SME's.

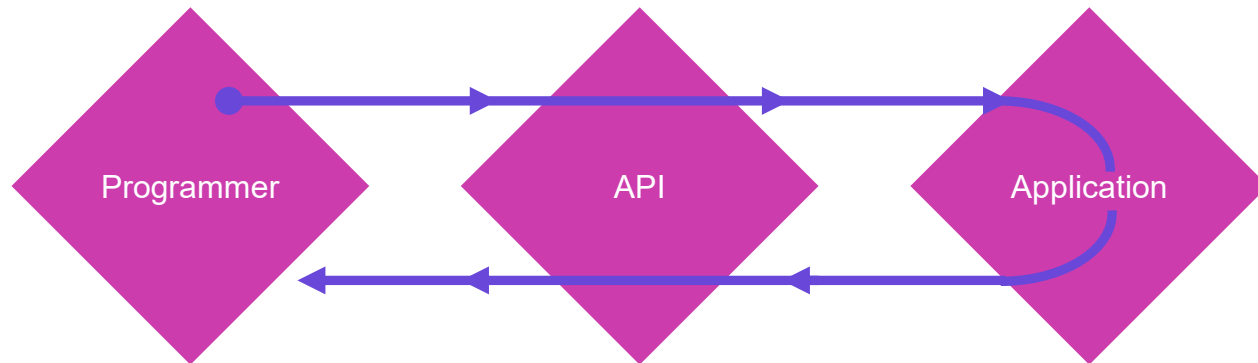
# INTRODUCTION TO APIs

# WHAT IS AN API?

## Application Programming Interface

*Functions and procedures that allow the creation of applications that access the features or data of an operating system, application, or other service.*

- Allows two applications to pass information
- HTTP protocols remove the “language” barrier
- Permissions and security enforced by the parent application (API author)



# API TYPES

## ➤ Web service APIs

- SOAP
- XML-RPC and JSON-RPC
- REST

## ➤ WebSocket APIs

## ➤ Library-based APIs

- JavaScript
- TWAIN

## ➤ Class-based APIs (object oriented)

- Java API
- Android API

## ➤ OS functions and routines

- Access to a file system
- Access to a user interface

## ➤ OS functions and routines

- Access to a file system
- Access to a user interface

## ➤ Object remoting APIs

- CORBA
- .NET remoting

## ➤ Hardware APIs

- Video acceleration
- Hard disk drives

## ➤ PCI buses

# WHERE IS OPEN API IN THAT LIST?

REST and SOAP are API specifications. Open API is the implementation.

- Specification - How the API functions (Rules)
  - Programming framework
  - Transmission protocols
  - Data format
- Implementation - How the API is published (Access)
  - Location of the definition (website, private)
  - User interface

What do you call the rules for my product's individual API?

**An Open API is publicly available and provides access to proprietary software through a URL.**

# SPECIFICATION OR DEFINITION

Sometimes, developers will erroneously call the rules of their API a specification.

## API Specification

A broad overview of functionality and expected results, typically shared by several APIs.

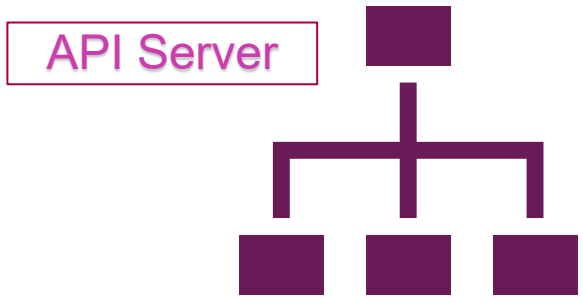
- Defines rules (framework) for designing an API.
- Includes how to organize URL paths, code parameters, and data models.

## API Definition

JSON objects that define the core settings for an individual API.

- Defines the actions for inbound and outbound calls.
- Includes comments that can be generated into documentation.

# HOW DO APIs COMMUNICATE?

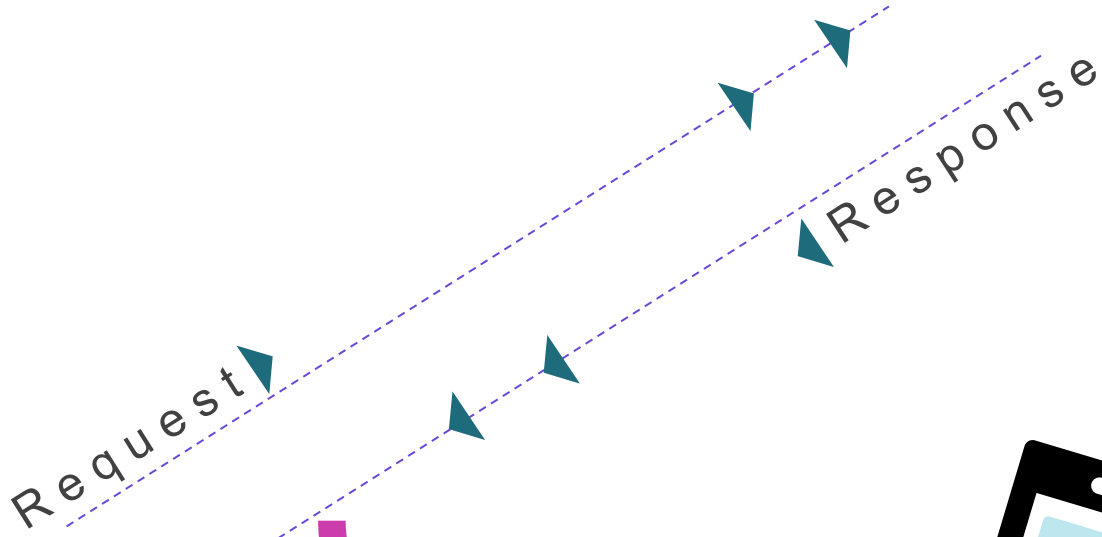


External Developer

POST <http://server-name.api.com/MortgageCheck>



```
[  
  {  
    "description": "quarter",  
    "mode": "REQUIRED",  
    "name": "qtr",  
    "type": "STRING"  
  },  
  {  
    "description": "sales representative",  
    "mode": "NULLABLE",  
    "name": "rep",  
    "type": "STRING"  
  },  
  {  
    "description": "total sales",  
    "mode": "REQUIRED",  
    "name": "total_sales",  
    "type": "INTEGER"  
  }  
]
```





# LEGO OF BUILDING ALL THE FEATURES

1. Use your Legos to create something.

This is your application.

- What does it do well?
- Are there features planned for future development?

2. Now, go around the room and add more Legos to your creation. You can add from anyone else's Legos, and you can add as many or as few as you like.

- Were you able to improve your design?
- Was the process more creative?

APIs are like these Legos. Your application can provide features (bricks) to other applications, and your application can consume data from other applications to:

- use existing features from other applications to make your product more robust.
- expand on your product's existing features by sharing information between applications.
- combine data between applications to create a completely new offering.
- add clients to your product without development overhead.



# PARTS OF AN API

# PARTS OF AN API REQUEST

## URL

Defines the location of an endpoint as a web address that tells the application what kind of data is being requested.

## METHOD

Describes how an endpoint interacts with the application.

## HEADER

Describes how an endpoint interacts with the application.

## REQUEST SCHEMA

(Optional) Passes request data to the parent application.

- GET
- PUT
- POST
- PATCH
- DELETE

# URLs

Defines the location of an endpoint as a web address that tells the application what kind of data is being requested.

- **Domain Name** - Server where the API is hosted
- **Resource Collections** - Logical groups of endpoints:
  - Deposit Accounts
  - Loan Accounts
  - Monetary Transactions
  - Customer Records
- **Endpoint Name** - Unique web location used to process the call
- **Path Parameter**- (optional) Identifies a unique record in the database
- **Query Parameter**- (optional) Filters or sorts the records returned in the response

# OPTIONAL PARAMETERS IN URLs

Most sort and filter options are passed as part of the URL, using a **Query Parameter**.

- Appear after a question mark (?) in the endpoint.
- Each parameter is listed one right after the other with an ampersand (&) separating them.
- The order of the parameters does not matter.

REQUEST METHOD	URL							
GET	<code>http://api.example.com/search?q=%22api%20monitoring%22&amp;lang=en</code>	Validate Step						
HEADERS	PARAMETERS	AUTHENTICATION						
	<table><tbody><tr><td>q</td><td>"api monitoring"</td><td>🗑️ +</td></tr><tr><td>lang</td><td>en</td><td>🗑️ +</td></tr></tbody></table>	q	"api monitoring"	🗑️ +	lang	en	🗑️ +	
q	"api monitoring"	🗑️ +						
lang	en	🗑️ +						

PATH PARAMETERS & QUERY PARAMETERS ARE PASSED IN THE URL

# URLs

Defines the location of an endpoint as a web address that tells the application what kind of data is being requested.

- **Domain Name** - Server where the API is hosted
- **Resource Collections** - Logical groups of endpoints:
  - Deposit Accounts
  - Loan Accounts
  - Monetary Transactions
  - Customer Records
- **Endpoint Name** - Unique web location used to process the call
- **Path Parameter**- (optional) Identifies a unique record in the database
- **Query Parameter**- (optional) Filters or sorts the records returned in the response

```
https://maps.googleapis.com/maps/api/js?key=APIKEY&callback=initMap
```

# METHOD

Describes how an endpoint interacts with the application.

- **GET** - Pulls data from the parent product
  - Cannot make changes
  - Filters can sometimes be included so only some records are returned
  - Think of this method as View Only
- **POST** - Adds a new record only
- **PATCH** - Updates an existing record only
- **PUT** - Usually updates data, but in some cases, can be coded to create records.
  - To create, the request must provide the resource id, such as an account number
- **DELETE** - Removes a record

# REQUEST HEADER

Describes how an endpoint interacts with the application.

## ➤ What is an HTTP header?

Everything in between the `<head>` `</head>` tags on a web page. The information contained the header tells the server receiving the call:

- the method,
- the path (URL), and
- the authorization (key) that the computer sending the call is using for access.

## ➤ How does the API use Header parameters?

- Typically used across all endpoints in the API.
- Sometimes, unique header parameters are created for one **resource collection**.

Headers can be defined for both **requests and responses** and are unique to your product.

Additional header information is passed, but it is not relevant to this discussion.



# HEADER PARAMETERS

The screenshot shows the Chrome DevTools Network panel with the 'Headers' tab selected. The 'Response Headers' section is expanded, showing various headers such as Date, Server, X-Powered-By, Pragma, Expires, Etag, Cache-Control, Content-Type, Last-Modified, X-Pingback, Content-Encoding, and Vary. The 'Request Headers' section is also expanded, showing Host, User-Agent, Accept, Accept-Language, Accept-Encoding, Accept-Charset, Keep-Alive, Connection, Cookie, If-Modified-Since, If-None-Match, and Cache-Control.

Method	Status	Modified	Domain
GET	304 Not Modified		net.tutspus.com
GET	304 Not Modified		net.tutspus.com
GET	304 Not Modified		net.tutspus.com
GET	304 Not Modified		ajax.googleapis.com
GET	304 Not Modified		net.tutspus.com

## HEADER PARAMETERS

Unique parameter for Phoenix only - allows for open ended filtering of some GET results.

This header parameter allows the sender to pass a unique id number. If empty, Phoenix creates one to track API calls.

Authorization

### X-PHX-Filters

string

You can filter your list to contain only items which match a simple logical expression using this header parameter.

**Syntax:** filter = (propertyName operator value)

**Operators:** lt (<), le (<=), gt (>), ga (>=), eq (=), ne (<>)

**Examples:**

X-PHX-Filters = (status eq 'Active') and (tin eq '123-84-9001')

X-PHX-Filters = (empId gt 0) or (status eq 'Active')

X-PHX-Filters = (status eq 'Closed') and (lastLogonDt gt '2004-10-15')

### X-PHX-ReferenceNo

string

Passes the reference number for the call.

The system will generate a unique reference number if not provided. When passed, uniqueness is not verified by the system, however, passing unique values is recommended. Its value will be returned in the Response as referenceNo.

### X-PHX-JWT

required

string

Passes the generated JWT token.

**Note:** You can use the Firebug extension in Firefox for a formatted view of the HTTP header.

# REQUEST SCHEMA (PARAMETERS)

(Optional) Passes data in request parameters to the parent application.

```
60
61 //This class is used just for receiving the request from endpoint.
62 //It will get mapped to GbCombStmntAddrTpiRequest which inturn will be used for further processing.
63 public class GbCombStmntAddrExt
64 {
65     #region Field AddressBasis
66     /// <summary>
67     /// {{GB_COMB_STMT_ADDR.Address_Basis}}
68     /// </summary>
69     [MaxLength(12)]
70     [Required()]
71     [DataType("Char(12)")]
72     public string AddressBasis { get; set; }
73 #endregion
74
75     #region Field EndDt
76     /// <summary>
77     /// {{GB_COMB_STMT_ADDR.End_Dt}}
78     /// </summary>
79     [DataType("SmallDateTime")]
80     public System.Nullable<System.DateTime> EndDt { get; set; }
81 #endregion
82
83     #region Field Hold
84     /// <summary>
85     /// {{GB_COMB_STMT_ADDR.Hold}}
86     /// </summary>
87     [MaxLength(1)]
88     [DataType("Char(1)")]
89     public string Hold { get; set; }
90 #endregion
91
92     #region Field IncludeImages
93     /// <summary>
94     /// {{GB_COMB_STMT_ADDR.Include_Images}}
95     /// </summary>
96     [MaxLength(1)]
97     [Required()]
98     [DataType("Char(1)")]
99     public string IncludeImages { get; set; }
```

How would I know if a call has Request parameters to pass in the Request Body?

# PARTS OF AN API RESPONSE

## STATUS CODE

Indicates whether the call was successful, and if not, provides information on why the call failed.

## RESPONSE HEADER

Same as the HTTP Header discussed above, but the data is being passed back from the server.

## RESPONSE BODY

Similar to the Request body, but the data is being passed back from the server.

# STATUS CODE

Indicates whether the call was successful, and if not, provides information on why the call failed.



HTTP Status Codes are already defined, but you may need to review the language used for custom error codes.

<https://httpstatus.com/>

# RESPONSE BODY

Similar to the Request Schema, but the data is being passed back from the API server.

The call response will only include Response Parameters when the endpoint has been designed to return field data from the product.

For example, when I perform a GET call to Retrieve 3<sup>rd</sup> Party Dealers, the response returns:

- A list of the names of all the 3<sup>rd</sup> party dealers saved in the Fusion Phoenix database.
- A description of the dealer type, such as Auto Dealer or Jewelry Dealer.
- The unique id number from the Dealer table that identifies the dealer.
- The unique id number from the RIM table that identifies the individual's record.

Each response is coded by the developer to return parameters (fields) that relate to the call being performed.

# RESPONSE BODY (PARAMETERS)

```
{
  - "messages": [
    - {
      "fieldName": "string",
      "msgId": 0,
      "msgText": "string",
      "msgType": "string",
      "xRefId": 0
    }
  ],
  - "pagingInfo": {
    "currentPage": 0,
    "moreRecords": "string",
    "pageSize": 0,
    "totalResults": 0
  },
  "referenceNo": "string",
  - "responseInfo": {
    "cvReturnCode": "string",
    "logPtid": 0,
    "postingMethod": "string",
    "primaryDbAvailable": "string",
    "returnCode": 0,
    "xmDbStatus": "string"
  },
  - "result": [
    - {
      "description": "string",
      "firstName": "string",
      "lastName": "string",
      "rim3rdDlrId": 0,
      "rimNo": 0
    }
  ],
  "statusCode": 0
}
```

**messages** provides any messages that the server produced while processing the call.

**pagingInfo** provides page data fields with context about the amount of fields that are returned from the database.

**responseInfo** object includes parameters that are returned for every call that relays technical details back to the sender, including the status of the server, a unique id for the call (logPtid), and confirmation of how the call was posted (real-time or future dated).

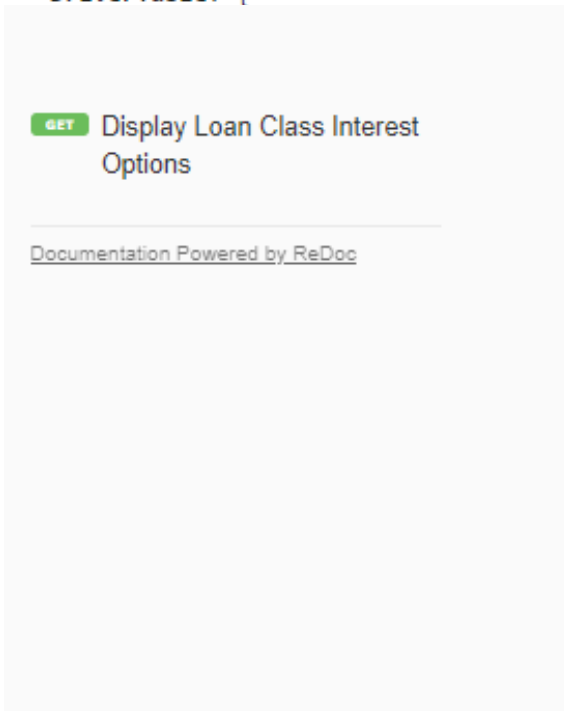
**Result** fields that clients typically see in the application. Returns the field values when a GET call is performed.

Note that the Status Code is returned as part of the response.

# RESPONSE BODY (PARAMETERS)

```
{  
- MRData: {  
  xmlns: "http://ergast.com/mrd/1.4",  
  series: "f1",  
  url: "http://ergast.com/api/f1/2016/drivers.json",  
  limit: "30",  
  offset: "0",  
  total: "23",  
- DriverTable: {
```

To the 3<sup>rd</sup> party developer, the Request and Response are both formatted in plain text JavaScript Object Notation (JSON).

 <p>GET Display Loan Class Interest Options</p> <p>Documentation Powered by ReDoc</p>	acctType	string Defines the account type of the loan as defined in 'AD_GB_ACCT_TYPE'
	advNoticeRequired	string Indicates whether or not an advance notice must be sent to the borrower prior to an interest rate increase.
	baseRate	number <double> Indicates the base loan rate for adjustable loans.
	capInstruction	string Defines how interest is to be capitalized. Statement - Interest is capitalized when the loan statement is created. Payment - Interest is capitalized on the loan's payment due date. Period - Interest is capitalized based on capitalization period specifications.
	capInt	string Indicates if accrued interest is to be added to the loan's principal balance, rather than generating a separate billing for interest.

```
  },  
- {  
  driverId: "button",  
  permanentNumber: "22",  
  code: "RIT"
```

1

2

3

4

5

6

7

# SPOT THE PARTS OF AN API



Using the post it notes provided, write the API component and stick it on the board.

Request  
Parameter

HTTP Header

Response  
Schema

Resource  
Collection

Path  
Parameter

Status  
Code

Endpoint URL

METHOD



# **API CONTENT OBJECTIVES**

**(SCOPE ANALYSIS)**

# API CONTENT OBJECTIVES

When planning API content, you should analyze the following topics with your team to determine the overall scope of the writing process:

- Audience - Who will use the content?
- Purpose - How will the audience use the content?
- Scope - What needs to be communicated?
- Deliverables - What content types need to be delivered?
- Schedule - What activities are required to complete the content?
- Dependencies - What risks have the potential to affect delivery?

# AUDIENCE

Define who will use API documentation.

- Internal Resources - Already familiar with the product
- Clients - Have access to the product and full support
- External Developers - Access only to the API

## PURPOSE

# POP QUIZ!

Identify the action each method performs.

**DELETE**

**GET**

**PUT**

**POST**

**PATCH**

● Update an existing record.

● Remove a record from the database.

● Create a new record.

● View records only.

● Create or Update records.

# SCOPE

Define the requirements for successful API implementation.

## Reference Content (API Library)

- Endpoint titles and descriptions
- Header parameter descriptions
- Query parameter descriptions
- Field (parameter) definitions for the Request and Response
- Object/Array descriptions

## Non-Reference Content (Help Topics)

- Clear, high-level summary of the API's purpose
- Setup and authentication information
- Well-defined Resource Collections
- Defined status and error codes
- Rate limiting and thresholds
- Code samples

# DELIVERABLES

Define the API documentation to be delivered

- API Overview
  - Summary and Getting Started
  - Authentication
  - Status and Error Codes
  - Rate Limiting and Thresholds
  - Code Samples (Tutorials)
- API reference library (the online user interface)
- Installation Guide (for closed APIs only)

# SCHEDULE

Estimate the Project

- How well defined are the project files? Can you locate them easily?
- Is there a list of Request fields for each endpoint? How about Response fields?
- Has the team documented which endpoints will include the path parameter?
- Had the team documented which header parameters will accompany each endpoint?
- Who is responsible for non-reference content and how much is currently complete?

If your development team is working in sprints and the final document is not clearly defined, it is imperative that you follow up in each sprint review to get a count of Titles, Descriptions, Parameters, and Status/Error Codes that need to be written or reviewed.

This helps ensure that there will not be scope creep that throws the project timeline off track.


# DEPENDENCIES

Define the needed resources and associated risks.

- API Overview (Non-Reference Content)
  - Summary and Getting Started - Demo (QA/Dev)
  - Authentication - Provided by architecture
  - Status and Error Codes - Requires Development documentation
    - May pull existing errors from the product, which are unfamiliar to external developers
  - Rate Limiting and Thresholds - Provided by architecture
  - Code Samples (Tutorials) - Provided by QA (reviewed by Product Analyst)
- API reference library (online user interface) - Provided primarily by PA and Doc
  - Requires research of existing fields and documentation already available to clients.
- Installation Guide (for closed APIs only) - Demo provided by QA/Dev
  - Requires testing and screenshots



# ARE APIs SELF DOCUMENTING?



“We’re adopting standard programming conventions and human-readable names so users can understand the system without prior specific knowledge.”

## Our Products Are Closed Source Software:

- Restricted access to business logic
- Traditional help files not included with an open API
- Support is paid for by application clients (No support for external developers)
- Not all code comments are exposed
- Proprietary developers may assume external developers share the same set of skills

Only when source code for the API is **open** can external developers review the code that controls what an API does.

## REFERENCE DOCUMENTATION TYPES

Knowledge Type	Description
Functionality and Behavior	Describes what the API does (or does not do) in terms of functionality or features. Describes what happens when the API is used (a field value is set, or a method is called).
Concepts	Explains the meaning of terms used to name or describe an API element, or describes design or domain concepts used or implemented by the API.
Directives	Specifies what users are allowed and not allowed to do with the API element. Directives are clear contracts.
Purpose and Rationale	Explains the purpose of providing an element or the rationale of a certain design decision. Typically, this is information that answers a "why" question: Why is this element provided by the API? Why is this designed this way? Why would we want to use this?
Quality Attributes and Internal Aspects	Describes quality attributes of the API, also known as non-functional requirements, for example, the performance implications. Also applies to information about the API's internal implementation that is only indirectly related to its observable behavior.
Control-Flow	Describes how the API (or the framework) manages the flow of control, for example by stating what events cause a certain callback to be triggered, or by listing the order in which API methods will be automatically called by the framework itself.

Knowledge Type	Description
Structure	Describes the internal organization of a compound element (e.g. important classes, fields, or methods), information about type hierarchies, or how elements are related to each other.
Patterns	Describes how to accomplish specific outcomes with the API, for example, how to implement a certain scenario, how the behavior of an element can be customized, etc.
Code Examples	Provides code examples of how to use and combine elements to implement certain functionality or design outcomes.
Environment	Describes aspects related to the environment in which the API is used, but not the API directly, e.g., compatibility issues, differences between versions, or licensing information.
References	Includes any pointer to external documents, either in the form of hyperlinks, tagged "see also" reference, or mentions of other documents (such as standards or manuals).
Non-information	A section of documentation containing any complete sentence or self-contained fragment of text that provides only uninformative boilerplate text.

# EXAMPLES OF NON-SELF DOCUMENTING CONTENT

## 1. Interactions between UI Elements

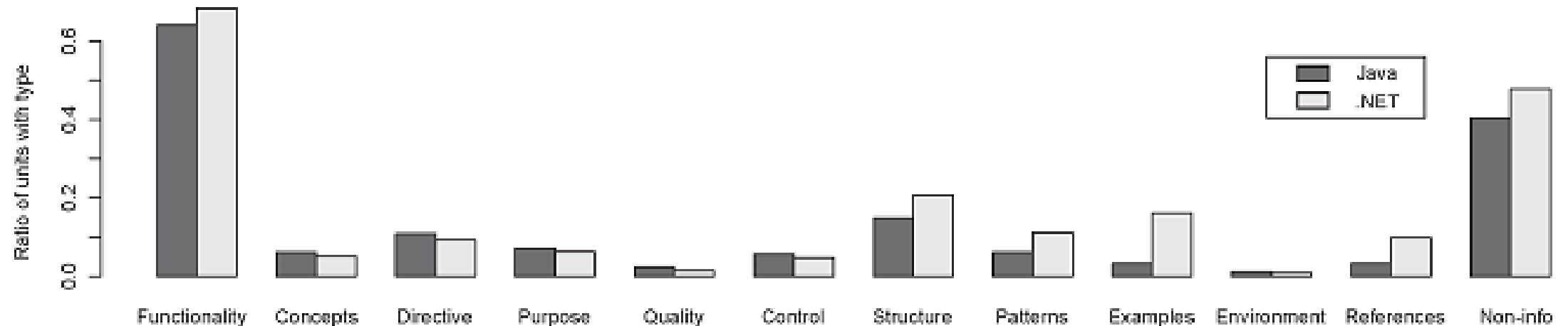
- If Percent is Y, this field is disabled.
- Simple cannot be selected when DirResType is Add-On.

## 2. Request/Response Constraints

- Provide either the account type or the application type for the request.
- Float settings cannot be updated when ItemType is sent with a request.

## 3. Coded Data in Tables

- 1 (Real-Time), 2 (Memo Post), 3 (Real-Time, Reject)



Distribution of knowledge types across API elements in the Java SDK 6 and .NET 4.0.

# APIs ARE NOT SELF-DOCUMENTING

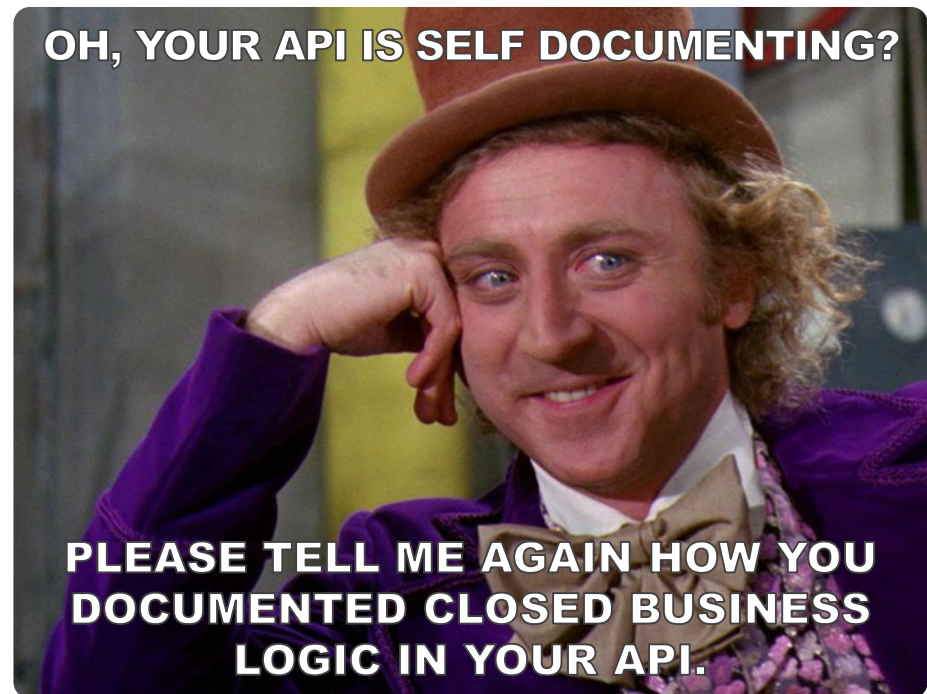
Self-documentation is still based the words contained in the code. While a developer may mean self-documenting as a way to procedurally generate help content, that does not preclude technical writers from performing the necessary work of ensuring adequate detail is provided, spelling and grammar is correct, and the content for the API clearly communicates what is and is not possible.

This is client-facing documentation, and for most external developers, it is the **only** documentation they will see from our company.

## API documentation affects our reputation

- Endpoint titles should follow a standard convention.
- Field (parameter) descriptions should be robust.
- Constraints and dependencies should be called out.
- Coded data must be revealed.

Ultimately, the writer determines what is seen by clients.



# REVIEW

# YESTERDAY'S TOPICS

## ➤ Introduction to APIs

What is an API?

Why develop APIs?

## ➤ Parts of an API

Parts of a Request

Parts of a Response

## ➤ Content Objectives

Audience

Purpose

Scope

Deliverables

Schedule

Dependencies

# DEMO FOR API CALLS USING POSTMAN

Download and Install Postman: <https://www.getpostman.com/>

## Demo:

1. Create a folder on your desktop and saved the attached files inside it.
2. Go to postman, and paste the URL for the API endpoint: <https://ghibliapi.herokuapp.com/films>
3. Double-click index.html to perform the call and get the translated response.
4. Studio Ghibli API: <https://ghibliapi.herokuapp.com/>

## Test Postman:

1. NASA API: <https://api.nasa.gov/index.html>
2. Get an API Key (will be sent to your email).
3. Select an API from NASA's list, and put your key in the Path Parameter.
4. Click Send to get the response.
5. Postman will automatically display available filters/sorts (parameters) in the UI.



# API STANDARDS GUIDE

# TITLE STANDARDS

Each endpoint title is unique and describes the action the API call performs.

The Method determines the first part of the title followed by the business function.

<b>METHOD</b>	<b>Title Starts with...</b>	<b>Include the Business Function</b>
GET	Retrieve <i>(Search for when filters are provided that are intended to return a specific record)</i>	Deposit Accounts by Account Type
POST	Create New	Hold Stop for Deposit Transaction
PATCH	Update	Teller Drawer Balance
DELETE	Delete	Correspondence Message

## Exceptions:

- Deposit Account **Origination** (POST)
- **Assign** User Defined Fields for Financial Statement (POST)

## DESCRIPTION STANDARDS

Each endpoint description provides a few sentences that provides the purpose and functionality of an endpoint. Be sure to include unique considerations or other technical details that might be required.

- Use complete sentences
- Always start the description with, “This call...”
- Again, the method determines the verbs used in the description.

**Note:** If updating legacy calls, the description may include the name of the equivalent call.

## METHOD

GET	This call <b>returns</b> a list of deposit accounts.
POST	This call <b>adds a new</b> hard hold or temporary stop payment to a deposit transaction.
PATCH	This call <b>updates</b> the drawer balance for a selected teller, which can be used to balance the teller journal during the closing process.
DELETE	This call <b>deletes</b> a selected email message from the system.

# DESCRIPTION STANDARDS

Provide as much detail as you would a window description.

## Create New Recurring Transfer

This call creates a new funds transfer that automatically recurs on a periodic basis between two deposit accounts or a deposit account and a line of credit (LOC) account.

**Availability:** Always available. This transaction can be processed real-time or memo posted.

**PhoenixXM Equivalent:** Update 10610 - Create New Auto Transfer

## Retrieve Customer/Member Account Classes

This call returns a list of customer/member account classes and their settings. Customer/Member classes are defined by the institution on the Add New/Edit Existing Customer/Member Class window in System Administration.

**Availability:** Always available.

## Update the Institution Rate Index

This call updates the rate index applied globally across the financial institution. Rates can be set to take effect immediately or at a future date.

**Availability:** Always available. This transaction is processed real-time or memo posted.

**PhoenixXM Equivalent:** Update 11045 - Output Type 1

# DESCRIPTION STANDARDS

## Questions for Subject Matter Experts

- What does the endpoint allow an external developer to do?
- Are there any restrictions to *when* the call can be performed?
- Are there restrictions for who (Employees Only) can perform the call?
- Is there a call that needs to be made before this call in order to have the data to perform this call?

For example, a call might require an external developer to call a list of existing addresses for a customer if attaching a statement to a new address is not allowed.

- For PUT, POST, and PATCH endpoints:
  - Are there constraints or dependencies in the fields that can be submitted in the call?
  -

# PARAMETER STANDARDS

Each parameter description should provide details of what is stored in the field. Most of the time the field name is recognizable, but it will not always be apparent to a 3<sup>rd</sup> party developer.

- Start with a capitalized, 3<sup>rd</sup> person, present tense verb
- For parameters that sort or filter, choose a verb that clearly indicates the purpose of the Query parameter
- The resulting description creates a complete sentence and eliminates repeating, “This field...”

**acctType** Indicates the account type that is assigned to an account.

**accumBasis** Indicates whether the number of times a charge has been assessed begins with the First Occurrence of the assessed charge or the Last Period.

**companyId** Defines the originator of a transaction. When the company id starts with:

- 1-Uses the IRS Employer Identification Number
- 3-Assigned by Data Universal Numbering Systems
- 9-Added manually by an employee

Review the window definitions in your product that correspond to the field in the API to ensure that all critical closed business logic is included in the definition.

***IMPORTANT!*** The same standards apply for all parameter types.

# PARAMETERS THAT SORT AND FILTER (HEADER/QUERY)

**fields** Limits which fields are returned in the response. When this field is blank, all fields are returned.

**Syntax:** fields = (Comma separated field names)

**Example:** fields = acctType, acctNo, amt

**branchNo** Filters results to return only records that were added from the selected branch.

**orderBy** Sorts the list returned by field name and ordering option.

**Syntax:** orderby = fieldName [ asc | desc ] [ ,...n ]

If no sort option provided, the default is ascending

**Examples:**

orderby = createDt - Results are sorted by the date created from oldest to newest (default).

orderby = emplId, createDt desc - Results are sorted first by emplId (ascending) and then createDt (descending).

**IMPORTANT!** The same standards apply for all parameter types.

# DOCUMENTING ENDPOINTS

In this activity, we are going to review and write the documentation for some endpoints. Review the information provided and compose the following:

- Summary of the Resource Collection
- Endpoint Title
- Description for the endpoint
- Description for the path parameter
- Description for header parameters
- Description for Request body parameters
- Description for Response body objects/arrays
- Description for Response body parameters



# DOCUMENTING ENDPOINTS

In this activity, we are going to review and write the documentation for some endpoints. Review the information provided and compose the following:

The screenshot displays the Swagger UI interface for an API. On the left, a sidebar lists endpoints under categories like PET, STORE, and USER. The main area shows the OpenAPI specification for the `POST /pet` endpoint. On the right, the UI provides a visual representation of this endpoint, including a 'Try it out' button, a table for parameters, and a JSON example for the body.

Red arrows indicate the following correspondences:

- From the OpenAPI spec's `paths: /pet:` to the UI's endpoint header `POST /pet`.
- From the spec's `summary: Add a new pet to the store` to the UI's description `Add a new pet to the store`.
- From the spec's `required: true` to the UI's `body * required` label.
- From the spec's `$ref: '#/definitions/Pet'` to the UI's JSON example for the body.

```
32 # - http
33 paths:
34   /pet:
35     post:
36       tags:
37         - pet
38       summary: Add a new pet to the store
39       operationId: addPet
40       consumes:
41         - application/json
42         - application/xml
43       produces:
44         - application/json
45         - application/xml
46       parameters:
47         - in: body
48           name: body
49           description: Pet object that needs to be added to the store
50           required: true
51           schema:
52             $ref: '#/definitions/Pet'
53       responses:
54         405:
55           description: Invalid input
56       security:
57         - petstore_auth:
58           - write:pets
59           - read:pets
60     put:
61       tags:
62         - pet
63       summary: Update an existing pet
64       operationId: updatePet
65       consumes:
66         - application/json
67         - application/xml
68       produces:
69         - application/json
70         - application/xml
71       parameters:
72         - in: body
73           name: body
```

**POST** /pet Add a new pet to the store

Parameters

Name	Description
body * required	Pet object that needs to be added to the store

(body)

Example Value | Model

```
{
  "id": 0,
  "category": {
    "id": 0,
    "name": "string"
  },
  "name": "doggie",
  "photoUrls": [
    "string"
  ],
  "tags": [
    {
      "id": 0,
      "name": "string"
    }
  ],
  "status": "available"
}
```

Parameter content type: application/json

Responses

Response content type: application/json

# **DOCUMENTING NON-REFERENCE CONTENT**

# DELIVERABLES

In addition to endpoints (reference content), Open APIs deliver non-reference content to help developers understand the functionality and organization of the calls.

- Overview
  - URL Structure
  - Resource Categories
- Getting Started (Quick Start)
- Request Schema Structure
- Response Body Structure
- Authentication and Authorization
- Technical Guidelines

# OVERVIEW

Provide a high level overview of what the API is to be used for.

- Purpose - What is it?
- Audience - Who is it for?
- Market Need - Why would a developer use it?
- Business Goals - Provide a few short scenarios that explain the market problems solved by your API.

An overview is the first chance to make a good impression.  
Keep the information focused on the audience need.



# OVERVIEW - URL (URI) STRUCTURE

While most APIs have a simple URI structure that can be understood easily, it can be helpful to outline the logic behind its organization.

- **Resource Categories** - Since the resource collection is part of the URL structure, you can also provide a summary of how endpoints are grouped, especially when the group name in the URI is not clear.



# GETTING STARTED

Detail the steps needed to start using the API (tutorial).

- Signing up for an account - Explain how to get credentials
- Getting API keys (procedure that summarizes Authentication and Authorization section)
- Making a Request - Walkthrough with sample of a request
- Evaluating a response - Provide the response to the sample request and include commentary on what is returned.

This is an end-to-end scenario that may use links to in-depth topics, but overall, you do not want to go in-depth in this section. The idea is to give an experienced developer a Quick Start guide, so they can skip any additional content they don't need.

<https://www.aerisweather.com/support/docs/api/getting-started/>

<https://developer.paypal.com/docs/api/overview/>

# REQUEST SCHEMA STRUCTURE

Explain each part of the Request Schema. Most importantly, this section should provide rules for making a request. Not all APIs have the same rules for posting!

Many APIs use this section to provide a dictionary of request parameters.

- Provide the Methods available.
- Path Parameters - Name and a description of how it works. For example, does it always require the same data or does the identifier vary?
- Header Parameters (Name/Description)
- Query Parameters (Name/Description)

Because parameters should be defined once and then referenced, a process should be created to generate a table of available parameters from a central location.

<https://docs.microsoft.com/en-us/previous-versions/office/office-365-api/api/version-2.0/batch-outlook-rest-requests>

<https://developer.paypal.com/docs/api/reference/api-requests/>

# RESPONSE BODY STRUCTURE

Explain each part of the Response Body.

- Sample of a response with commentary of unique characteristics
- Object/Array Descriptions - Some objects/arrays (groups of fields) may be consistently returned with data that was not requested but is required. These should be defined similar to how parameters were called out in the Request Schema section.
- HTTP Status Codes - If these are customized, explain how.
- Error Codes - Additional validation errors may be developed for the API, which usually return the HTTP status code 400 (Bad Request). This section should provide additional information that helps developers figure out why the call was rejected, which might be data that is not formatted correctly.

Additionally, if there is a debug process, that should be called out alongside errors.

<https://www.aerisweather.com/support/docs/api/getting-started/responses/>

<https://developer.paypal.com/docs/api/reference/api-responses/>



# AUTHENTICATION AND AUTHORIZATION

Provide a detailed guide to how the developer gets access to the API.

**Authentication** - The name of the workstation or user that is accessing a resource.

**Authorization** - The right to access a resource.

- What is the authentication method used?
  - HTTP (username/password)
  - API Keys (randomly generated hardware/IP data combo)
  - OAuth (JSON web token)
- Process of getting and maintaining access
- Structure of a token (if used), such as `header.payload.signature`
- Do keys/tokens expire? If so, provide expiration limits.

<https://developer.twitter.com/en/docs/basics/authentication/overview/oauth>

<https://www.dropbox.com/developers/reference/oauth-guide>

# TECHNICAL GUIDELINES

Additional development documentation is provided in this section.

- Best Practice - Recommended usage
- Rate Limit and Thresholds - The maximum number of calls (if defined by the API)
- Data Types - This was especially challenging, but not all developers use the same standards for data. It can be helpful to provide that information. Most users won't read it but when a call is failing repeatedly, this table can indicate the fix.
- Upgrade Considerations - When updates are made, what do existing developers need to know about how changes will be made?

<http://engqphxm3:18736/api-doc/#/api-doc/home>

<https://developers.google.com/maps/documentation/maps-static/dev-guide>

# TECHNICAL GUIDELINES

Additional development documentation is provided in this section.

- Code Samples - Typically, these are generated from the code. The most important part is that development provides commentary on **why** something is provided in the sample, not just the what.

Compare:

- **what:** “In this code, several arguments are passed to jQuery’s `ajax` method. The response is assigned to the data argument of the callback function, which in this case is `success`.”
  - **why:** “Use the `ajax` method from jQuery because it allows for asynchronous responses that won’t block the loading of your page.”
- FAQ, Troubleshooting, Support (Links should be visible from the homepage)

<http://engqphxm3:18736/api-doc/#/api-doc/home>

<https://developers.google.com/maps/documentation/maps-static/dev-guide>

# **CONTENT PLAN IMPLEMENTATION**

# PROJECT START

Schedule a first meeting with stakeholders to discuss the content plan:

## 1. SHOW A NEED

Have a scope of work ready to discuss. Use the terms from this workshop to explain the Reference Content and Non-Reference Content that should be delivered with a completed API.

## 2. MAKE THE ASK

Start with the primary ask - In order to complete this documentation project, the writer needs:

- Write access to the code to add/edit descriptions for endpoints and parameters.
- Training on check-in/check-out process.
- Demo of where endpoint structure is saved in the code (if not using Swagger).
- Web address of the API's user interface used by QA for testing.

## 3. DEMONSTRATE VALUE

- Present the better version of an endpoint by showing the code you would update. You can replicate this in a text editor if need be!
- Explain the alternative if they do not adopt this process:

Dev checks in code > QA reviews > Writer sends update in email >

Dev checks out code and makes all comment updates > Dev checks in code > QA reviews

# MEETING FOLLOW UP

After the first meeting, get results on the Asks.

## 1. Write access to the code to add/edit descriptions for endpoints and parameters.

Reiterate the importance. Show you are already prepared to make code edits.

## 2. Training on check-in/check-out process.

This can be completed by your closest development ally.

Do you need to download software, configure settings, or abide by certain rules? Anyone that is making code changes can provide this for you.

## 3. Demo of where endpoint structure is saved in the code (if not using Swagger)

Key questions for this training session:

- Where are endpoints saved?
- Where are the request schema saved?
- Where are the response bodies saved?
- For shared parameters (Path Parameters, Header Parameters, Query Parameters) -- is there a central file(s)?
- Will the writer be able to logically identify where components of an API are saved in the source code?

# UNIFYING THE TEAM UNDER A SINGLE GOAL

Reduce your workload by getting everyone aware of the content requirements for APIs

## 1. Present endpoint standards to the Product Analyst, Development, and Quality Assurance.

Highlight the language used in standards and the components that need to be coded to ensure all parts of the API are documented thoroughly.

## 2. Again, make the Ask clear.

- PA: Outline the components to be documented, and encourage PA's to follow the standards in Technical Specifications.
  - Dev: Request updates on parameters if they change during the development cycle.
  - QA: Request screenshots of Request and Response fields as part of the testing process.
- (optional) Perform standards checks as part of the QA process.

### Goal

Everyone involved in the Development Cycle is aware of what the Technical Writer will be doing **before** development even begins.

# UPDATE CONTENT AND REVIEW BEFORE RELEASE

## 1. Endpoints should be tracked by user stories.

Ensure you have all the parameters (path, header, request, response), and write the content to standards.

## 2. Make content updates (check in changes).

QA can be a helpful resource if you have questions about a parameter.

## 3. Review the API User Interface (UI) to confirm your changes published as expected.

You may need to know who (Build team) is tasked with updating the published UI.



# TOOLS FOR CODING AND DOCUMENTATION

FusionFabric.cloud

<https://www.fusionfabric.cloud/resources/developer-documentation>

Code, add documentation to the code, publish the API along with documentation from an all in one suite:

➤ Swagger - <https://swagger.io/>

Publish Non-reference content to FusionFabric.cloud

➤ R Project - <https://cloud.r-project.org/>

➤ R Studio - <https://www.rstudio.com/products/rstudio/download/>

➤ Bookdown - <https://bookdown.org/>

# ACTION ITEMS

## Individual Responsibilities

- › Learn markup language to format content
- › Learn basic HTML for any formatting that is unsupported by markup
- › Recognize the difference between comments and code
- › Training in the check-out/check-in process
- › Write and review Reference Content to align with standards

## Team Development

- › Generate documentation from code
- › Provide code Edit rights to technical writer
- › Orientation/training on file/naming hierarchy and structure
- › User interface output (publishing) and provide iterative access of this output to the writer

## Enterprise Development

- › Adoption of API standards for reference content across all products
- › Enforce documentation requirements for all endpoints brought in to FusionFabric.cloud

# ADDITIONAL RESOURCES

Eric Holscher ([ericholscher.com](http://ericholscher.com))

Co-founder of [Read the Docs](#) and [Write the Docs](#), where he works to elevate the status of documentation in the software industry.

Tom Johnson ([idratherbewriting.com](http://idratherbewriting.com))

Technical writer at Amazon in the Appstore group, focusing on third-party development of apps on Amazon devices (Fire TV and tablets).

## Reference Documents

OpenAPI Specification ([Swagger](#))

Patterns of Knowledge in API Reference Documentation ([Martin Robillard](#))

# Thank you

**Sarah Kittrell**  
Technical Writer

