ICT Project Guidance

Design:   
Technical - API Protocols & Extensions

Version:

0.2

Author:

Sky Sigal, Solution Architect

## Description

This document provides guidance as to which API protocol to use for public APIs versus private APIs, along with implementation guidance.

## Synopsis

Public APIs should be implemented as RESTful Resource based APIs over HTTP/S with OData and potentially also adding GraphQL interfaces. Remote Procedure Call based APIs should be reserved for Private APIs.

## Contents

[Description 1](#_Toc153714496)

[Synopsis 1](#_Toc153714497)

[Contents 2](#_Toc153714498)

[Introduction 4](#_Toc153714499)

[Risks 4](#_Toc153714500)

[Resolution 4](#_Toc153714501)

[Interface Types 4](#_Toc153714502)

[API Category Type 5](#_Toc153714503)

[API Protocols 5](#_Toc153714504)

[Web API Protocol Types 6](#_Toc153714505)

[Procedure Based APIs 6](#_Toc153714506)

[Resource Based APIs 6](#_Toc153714507)

[Use Cases 7](#_Toc153714508)

[Implementations 7](#_Toc153714509)

[Endpoints 7](#_Toc153714510)

[REST 7](#_Toc153714511)

[HTTP Verb 7](#_Toc153714512)

[Endpoint Naming 8](#_Toc153714513)

[Variables 8](#_Toc153714514)

[HTTP Response Codes 8](#_Toc153714515)

[ODATA 9](#_Toc153714516)

[Filtering 9](#_Toc153714517)

[Expand 10](#_Toc153714518)

[Select 10](#_Toc153714519)

[OrderBy 10](#_Toc153714520)

[Top & Skip 10](#_Toc153714521)

[Count 10](#_Toc153714522)

[GraphQL 11](#_Toc153714523)

[SOAP 11](#_Toc153714524)

[Recommendations 13](#_Toc153714525)

[Protocol Choice 13](#_Toc153714526)

[Terms & Conditions 13](#_Toc153714527)

[Requirements 14](#_Toc153714528)

[Transitional Requirement 14](#_Toc153714529)

[Quality Requirements 14](#_Toc153714530)

[Appendices 15](#_Toc153714531)

[Appendix A - Document Information 15](#_Toc153714532)

[Versions 15](#_Toc153714533)

[Images 15](#_Toc153714534)

[Tables 15](#_Toc153714535)

[References 15](#_Toc153714536)

[Review Distribution 15](#_Toc153714537)

[Audience 15](#_Toc153714538)

[Structure 15](#_Toc153714539)

[Diagrams 16](#_Toc153714540)

[Terms 16](#_Toc153714541)

## Introduction

While it is current best practice that systems be developed following API-First design principles, most applications are still being developed with integrated server-side generated user interfaces permitting developers to skip developing User Interface backing APIs, and almost never provide integration APIs.

## Risks

A system that does not provide a public API cannot be integrated with, hence contribute to a digital environment to supports a business becoming more efficient and responsive to its customers.

## Resolution

Systems must be developed with APIs that are appropriate to fulfilling current and envisioned integration task with the highest chance of remaining maintainable, performant and useful to other systems.

This document provides guidance on how to choose the most appropriate protocol for public APIs, along with high level guidance on expectations on how they are to be implement.

## Interface Types

An Application Programming Interface (API) provides a way for developers to access the functionality of service.

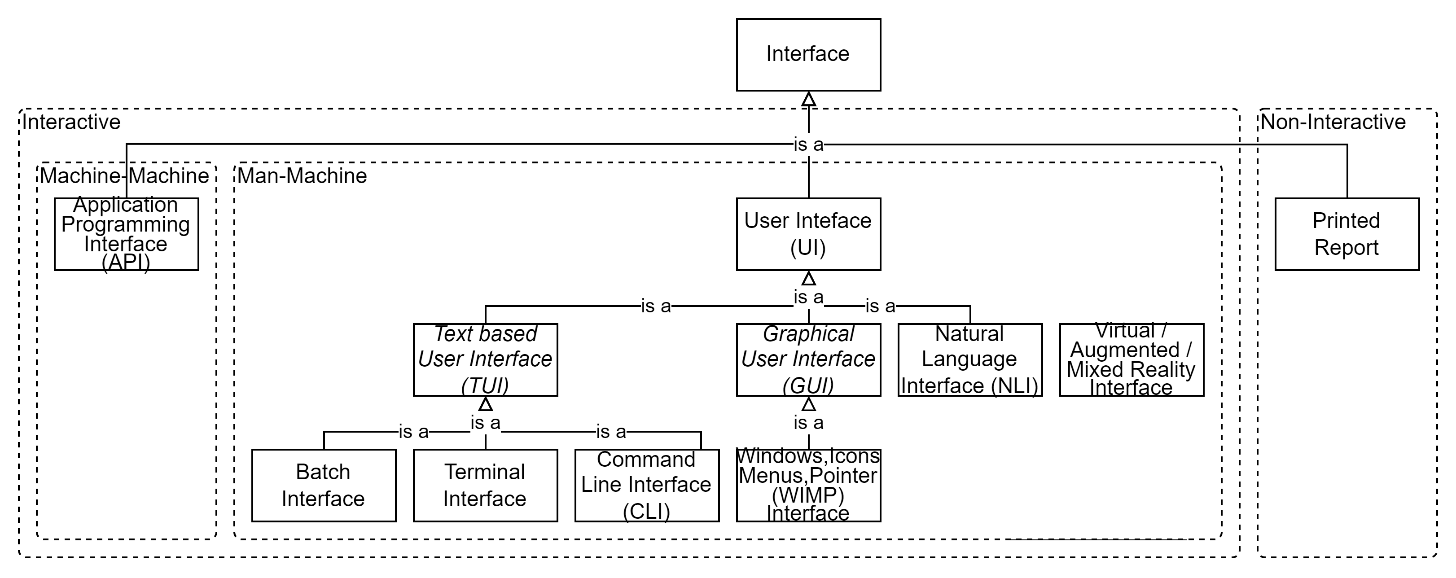


Figure : System Interface Types

As per Figure 1, APIs are but one of the types of interfaces commonly available for systems.

## API Category Type



Figure : API Purpose Classification

APIs are classifiable as being for Private, Partner or Public access.

Private APIs are consumed within an organization and not intended to be exposed to consumers outside of the organization.

Partner API are consumed by partners of your organization, working within an established relationship and contractual framework.

Public API are consumed by 3rd parties as a free or paid service. They can be Open[[1]](#footnote-2) (not require authentication) or require authentication[[2]](#footnote-3), Paid for or offered Free.

Note:  
Both Public and Partner APIs must have terms and conditions associated with their use.

## API Protocols

Several API protocols exist to choose from.

The location of intended consumers constrains the choice of protocol to use: same process, same device, same network, or another network.

Historically, on a same device, inter-process/same device communication on Windows used Common Object Model (COM), and within a network, Distributed Component Object Model (DCOM)[[3]](#footnote-4) or Common Object Request Broker Architecture (CORBA)[[4]](#footnote-5) were common options to consider.

The IT domain has since standardised for the most part on:

* building services from web-based components installed on different devices in different networks (instead of installing everything on the same device) -- thereby removing the need for both inter-process communication and within the same network inter-device communication over proprietary ports, and
* only using HTTP/S based protocols[[5]](#footnote-6) for all inter-device communication, even within the same network.

Several different communication protocols are available for use over an HTTP/S channel:

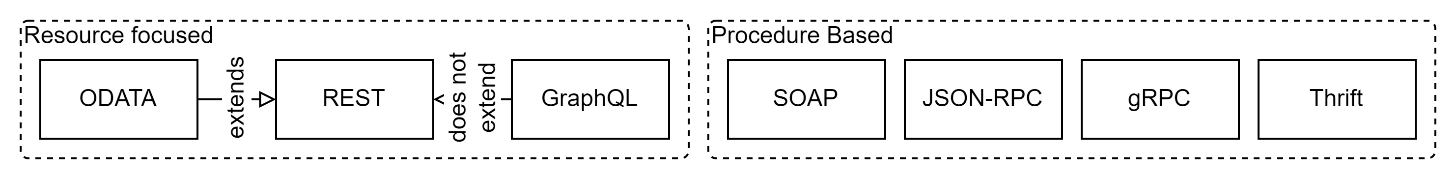


Figure : HTTP capable API Protocol`s

### Web API Protocol Types

The different API protocols generally fall into one of two broad categories, either focused on *resources* or focused on *procedures*. Each has logical advantages and disadvantages.

#### Procedure Based APIs

Procedure based APIs provide the most flexibility, at the cost of maintainability.

They make it very simple and direct for programmers to write a procedure in one program and call it from another. This is one of the characteristics that makes RPC so popular. But they also make it easy for technology and use-case assumptions to flow easily from one application to the other, thereby coupling the two and making the overall solution brittle.

Additionally, due to their ability to be used to expose a myriad of methods and objects, they require the most investment to develop, validate, secure, test, document, and support.

Finally, due to the variety of procedures and objects that can be exposed via RPC, they also require the most investment in learning from service consumers.

#### Resource Based APIs

On the other hand, resource-based APIs reduce the investment required by either party, focusing on providing access to Resources via a restrained number of HTTP operations (approximately 4).

Anyone who knows HTTP will be able to use the API, with no more documentation than a description of the API's entity model and a little bit of query syntax.

Importantly, HTTP/REST helps break that flow of assumptions, by forcing an intermediate translation from implementation procedures to an entity model. Procedures are no longer exposed directly at the interface to be called remotely; instead, API developers construct an entity model in between that disconnects the two sides. The entity-oriented model is not arbitrary; it is the conceptual data model of the problem domain as viewed by the client. It will likely have some relationship to an underlying storage data model but is usually simpler and more abstract. How effectively the API entity model decouples the caller from the callee depends a lot on the skill of the model’s designers, but the mere presence of the translation layer increases the chances of meaningful decoupling.

When an API is realized as an entity model, there is less tendency for unbridled organic growth, because entity models typically have a greater degree of coherence and overall structure.  Evolving an API based on an entity model requires explicitly adding a new type, property, or relationship to the model, which typically forces thought about how the addition fits with the overall model.

### Use Cases

For public APIs -- although slower than gRPC or Thrift -- Resource based protocols are recommended (e.g.: REST, or REST w/OData).

For partner APIs, consider offering REST, or REST w/ODATA and potentially GraphQL.

For private APIs – Procedure based (e.g.: gRPC-Web) is an alternative option to REST[[6]](#footnote-7).

Note:  
SPA interface APIs would be considered private APIs.

## Implementations

The following provide high level information regarding implementing Resource based public APIs.

### Endpoints

In alignment with the domain naming guidance provided elsewhere, consider providing each public API type under its own subset of the ‘api.’ Name:

* <https://rest.api.ourservice.ourorg.tld>
* <https://odata.api.ourservice.ourorg.tld>
* <https://graph.api.ourservice.ourorg.tld>

### REST

Based on the above observation, the rest of this document will concentrate on implementing APIs using Representational Stateless State Transfer (REST), and related extensions.

#### HTTP Verb

REST Endpoints are accessed with HTTP Verb:

* GET <https://rest.api.ourservice.ourorg.tld/persons>: returns 0+ existing resources.
* PUT <https://rest.api.ourservice.ourorg.tld/persons>: updates a single existing resource.
* POST <https://rest.api.ourservice.ourorg.tld/persons>: creates a new resource.
* DELETE <https://rest.api.ourservice.ourorg.tld/persons>: *logically* deletes a resource.

#### Endpoint Naming

By convention, REST endpoints are defined in the plural, using criteria to narrow the collection to a single entity. For example:

GET <https://rest.api.ourservice.ourorg.tld/persons> returns the list of Persons.

GET <https://rest.api.ourservice.ourorg.tld/persons/3414> returns a single Person with a system Id of 3414.

### Variables

While seminal, Fielding’s dissertation[[7]](#footnote-8) on REST is incomplete. While it describes how to use HTTP Verbs to retrieve serialised Resources it does not describe how to do the following actions.

* Serialisation format to use for responses (JSON, XML, etc.)
* Filter data (e.g.: select Persons who have blue eyes)
* Define shape and structure of the responses (e.g.: return just Id, Surname and Age, from a Person object)
* Ordering (e.g.: ordering list of responses by DOB, even if not returned)
* Paging (e.g.: return 3rd page of 20 persons)

This is addressed later using either OData or GraphQL.

### HTTP Response Codes

The response codes that RESTful APIs return are HTTP Response codes:

* 1xx Status Codes [Informational]
* 2xx Status Codes [Success]
* 3xx Status Codes [Redirection]
* 4xx Status Codes (Client Error)
* 5xx Status Codes (Server Error)

The more common response to HTTP Verbs will be:

* [All Verbs]:
  + 500 Internal Server Error
  + 503 Service Unavailable
  + 501 Not Implemented: endpoint not found.
  + 400 Bad Request: incorrect syntax.
  + 405 Method Not Allowed: the endpoint is known but disabled.
  + 401 Unauthorised: the requestor is not authenticated.
  + 403 Forbidden: the requestor is authenticated, but not authorised.
  + 202 Accepted
* GET
  + 200 OK
  + 410 Gone: resource no longer available.
* POST
  + 201 Created
  + 204 No Content
  + 409 Conflict: fails validation.
* PUT:
  + 200 OK
  + 204 No Content
  + 409 Conflict: fails validation.
* DELETE
  + 204 No Content
  + 200 OK 410 Gone: resource no longer available.

## ODATA

Maintainability of systems is decreased when developers design and implement novel solutions to solved standard problems. Hence it is recommended to extend REST using *ISO-20802 OData 4.0 Core Protocol* functionality.

Note:  
As per guidance given regarding Domain Naming conventions, the OData API should be made available as a subset of ‘api.’, parallel to ‘rest.’.

### Filtering

The $filter system query option allows clients to filter a collection of resources that are addressed by a request URL.

* https://odata.api.service.org.tld/Persons?$filter=Name eq 'Smith'
* https://odata.api.service.org.tld/Persons?$filter=Name ne 'Smith'
* https://odata.api.service.org.tld/Persons?$filter=Name gt 'Smith'
* https://odata.api.service.org.tld/Persons?$filter=Name ge 'Smith'
* https://odata.api.service.org.tld/Persons?$filter=Name lt 'Smith'
* https://odata.api.service.org.tld/Persons?$filter=Name le 'Smith'
* https://odata.api.service.org.tld/Persons?$filter=Name eq 'Smith' and Age lt 25
* https://odata.api.service.org.tld/Persons?$filter=Name eq 'Smith' or Age lt 25
* https://odata.api.service.org.tld/Persons?$filter=not endswith(Name,'ith')
* nbv0063https://odata.api.service.org.tld/Persons?$filter=style has Sales.Pattern'Yellow'
* https://odata.api.service.org.tld/Persons?$filter=Name in ('Smith', 'Campbell')

### Expand

The $expand system query option specifies the related resources or media streams to be included in line with retrieved resources. Each expandItem is evaluated relative to the entity containing the navigation or stream property being expanded.

* http://odata.api.service.org.tld/Persons?$expand=Category
* [http://odata.api.service.org.tld/Customers?$expand=Addresses/Country](http://host/service/Customers?$expand=Addresses/Country)

### Select

The $select system query option allows clients to request a specific set of properties for each entity or complex type.

The $select query option is often used in conjunction with the $expand system query option, to define the extent of the resource graph to return ($expand) and then specify a subset of properties for each resource in the graph ($select).

* [http://odata.api.service.org.tld/Persons?$select=Rating,ReleaseDate](http://host/service/Products?$select=Rating,ReleaseDate)
* [http://odata.api.service.org.tld/Persons?$select=\*](http://host/service/Products?$select=*)

### OrderBy

The $orderby system query option allows clients to request resources in a particular order.

* GET https://odata.api.service.org.tld/Persons?$orderby=DOB asc, IQ desc
* GET https://odata.api.service.org.tld/Categories?$expand=Persons($orderby=DOB asc, IQ desc)
* GET https://odata.api.service.org.tld/Categories?$orderby=Persons/$count

### Top & Skip

* [https://odata.api.service.org.tld/Persons?$top=10](http://host/service/Products?$top=10)
* [https://odata.api.service.org.tld/Persons?$skip=30&$top=10](http://host/service/Products?$skip=30&$top=10)

### Count

* <https://odata.api.service.org.tld/Persons?$count=true>
* https://odata.api.service.org.tld/Categories?$expand=Persons($count=true)

## GraphQL

GraphQL is an open standard that provides functionality in the same domain as ODATA, with many proponents outside of the Microsoft development space seem to prefer it to OData.

However, it is not ISO ratified, for a number of reasons, including the fact that -- unlike OData – its architecture is not compliant to the REST methodology.

Therefore, as government agencies are mandated to follow international Standards when they are available, but developers do like GraphQL, it should be implemented – but only after implementing OData first.

## SOAP

We recommend not offering a SOAP interface.

The protocol is an RPC based protocol. RPC based protocols should be reserved for private or partner APIs, for which there are more current protocols.

# Recommendations

## Protocol Choice

If the API’s purpose is to enable communication between two distributed components that you both own and control, and processing efficiency is a major concern, then a procedure-based API[[8]](#footnote-9) might be the most valuable choice.

However, if the primary objective is to make the system accessible to internal or external 3rd parties, focus on delivering an API that only requires a description of the APIs entity model, relying on well documented open standards[[9]](#footnote-10) for as much as possible.

## Terms & Conditions

Public and Partner APIs must have terms and conditions associated with their use. Key considerations are defining their behaviour (e.g.: throttling requests), requiring they update their service client to the latest major version within a specified duration, and the organisation’s commitment that they will provide the older major version for the same duration.

## Anti Patterns

### Too Specific Models

APIs transfer back and forth Data Transfer Objects (DTOs). DTOs are generally Plain Old Class Objects (POCOs) that have properties, but devoid of logic (as logic cannot be serialised over the wire, only properties can).

But it is important that each DTO developed requires testing, presumably first by static methods, then by dynamic behaviour testing done later.

Note:  
Testing effort can be significantly reduced if the DTOs inherit from common interfaces (IHasGuidId, IHasState, IHasIsApplicableFromToUtcDateTime, IHasNameAndDescription, etc.), such that testing can be abstractly developed against instances of classes that implement the interfaces rather than be custom made for each DTO, testing (ie cost) must be done, non the less.

Instead of making too specific models (eg: Student, Teacher, Parent), consider using more abstract POCOs (eg: Person). Not only is testing only required once for 3 different use cases, but it is actually more correct to develop Persons as Roles associated to Groups than as 3 different types.

### Variable Object Returns

OData can *$Select* specific properties from an attribute to project an abstract object. This has a valid use case.

Whereas it is incorrect to develop DTOs to be returned in two shapes, possibly one for Integration and one for user interface reasons. This is mixing up interface purpose (integration and user interface).

### Exposing Datastore Identifiers

It’s practically impossible to develop DTOs that don’t have an ID property to round trip the datastore Identity value, as is needed when updating resources.

But it is a security leak to expose the value within it such that it provides an attacker with valuable information.

An acceptable mitigation is to XOR the identifier with a known value, and XOR the responses. The mitigation is acceptable as it doesn’t just obfuscate, but encrypts the data such that it is unusable without knowledge of the unpublished known value.

Note:   
The known value should be regularly rotated (keeping the past one for a moment longer than normally needed), such as when the system is deployed, and unavailable to end users.

# Requirements

## Transitional Requirement

* The text of a versioned common base Terms and Conditions for API consumers must be developed.
* The Terms & Conditions for API consumers must be added to project’s publicly accessible website.
* A project specific website or corporate website section must be developed.
* The project must develop a project specific Domain Naming strategy.
* The service must deliver a REST based public API (e.g.: rest.api.ourservice.ourorg.tld)
* The service must deliver an ODATA extended REST based public API at a different endpoint than the REST endpoint (e.g.: odata.api.service.org.tld).

## Quality Requirements

* The URL used for API endpoints (e.g.: rest.api.myservice.myorg.tld) must be discoverable, made so by complying with the project’s agreed Domain Naming strategy.

Appendices

Appendix A - Document Information

### Versions

* 1. Initial Draft
  2. Reformatting

### Images

[Figure 1: System Interfaces Types 3](#_Toc153551368)

[Figure 2: API Purpose Classification 3](#_Toc153551369)

[Figure 3: HTTP capable API Protocol`s 5](#_Toc153551370)

### Tables

### References

There are no sources in the current document.

### Review Distribution

The document was distributed for review as below:

|  |  |
| --- | --- |
| Identity | Notes |
|  |  |
|  |  |
|  |  |

### Audience

The document is technical in nature, but parts are expected to be read and/or validated by a non-technical audience.

### Structure

Where possible, the document structure is guided by either ISO-\* standards or best practice.

### Diagrams

Diagrams are developed for a wide audience. Unless specifically for a technical audience, where the use of industry standard diagram types (ArchiMate, UML, C4), is appropriate, diagrams are developed as simple “box & line” monochrome diagrams.

### Terms

Refer to the project’s Glossary.

##### COM

: acronym for *Common Object Model*.

##### Common Object Model (COM)

: protocol for APIs within a Windows LAN network.

##### Common Object Request Broker Model

:…

##### CORBA

: acronym for *Common Object Request Broker Model*.

##### Distributed Object Model (DCOM)

: …

##### ICT

: acronym for Information & Communication Technology, the domain of defining Information elements and using technology to automate their communication between entities. IT is a subset of ICT.

##### IT

: acronym for Information, using Technology to automate and facilitate its management.

##### ODATA

: an ISO standard for extending REST applications with the ability to query, filter, project, page data in a standard manner, removing the need for the development of novel solutions.

1. We recommend that no API is Open to anonymous users. [↑](#footnote-ref-2)
2. By API Key, OAuth or registration. [↑](#footnote-ref-3)
3. Using a randomly selected TCP port between 49152 and 65535 [↑](#footnote-ref-4)
4. Using TCP/UDP 683/684/1050/2809 [↑](#footnote-ref-5)
5. over ports 80/443 [↑](#footnote-ref-6)
6. [Use gRPC in browser apps | Microsoft Learn](https://learn.microsoft.com/en-us/aspnet/core/grpc/browser?view=aspnetcore-8.0) [↑](#footnote-ref-7)
7. [Untitled Document (uci.edu)](https://ics.uci.edu/~fielding/pubs/dissertation/fielding_dissertation.pdf) [↑](#footnote-ref-8)
8. gRPC in particular [↑](#footnote-ref-9)
9. HTTP, JSON and optionally a standards-based Query syntax (e.g.: ODATA) [↑](#footnote-ref-10)