TECHNICAL SPECIFICATION

ISO/TS 23029

First edition
2020-02

Web-service-based application programming interface (WAPI) in financial services

Reference number
ISO/TS 23029:2020(E)

ISO copyright office
CP 401 • Ch. de Blandonnet 8
CH-1214 Vernier, Geneva
Phone: +41 22 749 01 11
Fax: +41 22 749 09 47
Email: copyright@iso.org
Website: www.iso.org

Published in Switzerland

Contents

Page

Foreword

ISO (the International Organization for Standardization) is a worldwide federation of national standards bodies (ISO member bodies). The work of preparing International Standards is normally carried out through ISO technical committees. Each member body interested in a subject for which a technical committee has been established has the right to be represented on that committee. International organizations, governmental and non-governmental, in liaison with ISO, also take part in the work. ISO collaborates closely with the International Electrotechnical Commission (IEC) on all matters of electrotechnical standardization.

The procedures used to develop this document and those intended for its further maintenance are described in the ISO/IEC Directives, Part 1. In particular, the different approval criteria needed for the different types of ISO documents should be noted. This document was drafted in accordance with the editorial rules of the ISO/IEC Directives, Part 2 (see www.iso.org/directives).

Attention is drawn to the possibility that some of the elements of this document may be the subject of patent rights. ISO shall not be held responsible for identifying any or all such patent rights. Details of any patent rights identified during the development of the document will be in the Introduction and/or on the ISO list of patent declarations received (see www.iso.org/patents).

Any trade name used in this document is information given for the convenience of users and does not constitute an endorsement.

For an explanation of the voluntary nature of standards, the meaning of ISO specific terms and expressions related to conformity assessment, as well as information about ISO's adherence to the World Trade Organization (WTO) principles in the Technical Barriers to Trade (TBT), see www.iso.org/iso/foreword.html.

This document was prepared by Technical Committee ISO/TC 68, *Financial services*, Subcommittee SC 9, *Information exchange for financial services*.

Any feedback or questions on this document should be directed to the user's national standards body. A complete listing of these bodies can be found at www.iso.org/members.html.

Introduction

0.1 Opening comments

The purpose of this document is to help implementers in the financial services industry define the framework, function and protocols for an application programming interface (API) ecosystem, enabling online synchronised interactions. It documents an international view of an API ecosystem in response to an urgent and significant world-wide demand for guidance and standardisation of APIs in financial services.

This has been driven by a number of emerging requirements – market-, corporate- and regulatory-driven – from different communities and jurisdictions for financial institutions to share data and enable functionality, such as between third parties acting on behalf of the customer, client or end user; between business to business processes; and within internal processes. It has been widely agreed that standardised APIs provide the most secure, developer-friendly, efficient, technically proven way of meeting many of these requirements. Moreover, it is understood that a standardised approach would unlock benefits conducive to promoting interoperability, enhancing security and supporting innovation. The sharing of data, and the subsequent use of APIs, is not limited to exchanges referenced in this document.

Despite these emerging requirements, there is currently no standardised approach at an international level. Moreover, there is no informative documentation covering the various considerations for developing APIs in financial services, especially given the maturity of some of its components (e.g. some are in draft). This document has been developed in response to meet these current and foreseen requirements that exist in the market. This document does not specify implementations of APIs, but instead takes an international view and references, as appropriate, specific implementation scenarios for illustrative purposes for guidance.

0.2 How to approach this document

This document should be approached as a best-practice framework for developing APIs in financial services. In this sense, some aspects of the document are more mature than others. The document is prescriptive where there is room to be. Where areas are less mature, commentary on best practice has been provided and the considerations set out.

Broadly speaking, this document adopts the following logic and order:

— Clause 3: terms and definitions used in the document;

— Clause 4: the initial considerations for the design of the API;

— Clause 5: overview and commentary on the different technology options;

— Clause 8: specific guidance on APIs under WAPI technical specification.

In Annex A, we set out an example of how to approach the document depending on a specific business area/desired API functionality.

Web-service-based application programming interface (WAPI) in financial services

1 Scope

This document defines the framework, function and protocols for an API ecosystem that will enable online synchronised interaction. Specifically, the document:

— defines a logical and technical layered approach for developing APIs, including transformational rules. Specific logical models (such as ISO 20022 models) are not included, but they will be referenced in the context of specific scenarios for guidance purposes;

— will primarily be thought about from a RESTful design point of view, but will consider alternative architectural styles (such as WebSocket and Webhook) where other blueprints or scenarios are offered;

— defines for the API ecosystem design principles of an API, rules of a Web-service-based API, the data payload and version control;

— sets out considerations relevant to security, identity and registration of an API ecosystem. Specific technical solutions will not be defined, but they will be referenced in the context of specific scenarios for guidance purposes;

— defines architectural usage beyond query/response asynchronous messaging towards publish/subscribe to support advanced and existing business models.

This document does not include:

— a specific technical specification of an API implementation in financial services;

— the development of JSON APIs based on the ISO 20022 specific message formats, such as PAIN, CAMT and PACS;

— a technical specification that is defined or determined by specific legal frameworks.

2 Normative references

There are no normative references in this document.

3 Terms and definitions

For the purposes of this document, the following terms and definitions apply.

ISO and IEC maintain terminological databases for use in standardization at the following addresses:

— ISO Online browsing platform: available at https://www.iso.org/obp

— IEC Electropedia: available at http://www.electropedia.org/

3.1
application programming interface
API
set of well-defined methods, functions, protocols, routines or commands which application software uses with facilities of programming languages to invoke services

Note 1 to entry: An API is available for different types of software, including Web-based system/ecosystem.

3.2
idempotency
idempotence
operation feature which means there is no additional effect if the operation is called more than once

Note 1 to entry: Idempotent operations are often used to design a Web-based system since it is hard to restrict redundancy access.

3.3
JavaScript object notation
JSON
open and text-based exchange format

Note 1 to entry: Data transmitted in JSON formats make it easy to read and write (for humans), parse and generate (for computers).

3.4
representational state transfer
REST
RESTful
software architectural style for distributed hypermedia systems Note 1 to entry: It was introduced and defined in 2000 by Roy Fielding in his doctoral dissertation.

3.5
resource hop
resource locator built on resource type and identifier, mandatory except for the last resource

Note 1 to entry: A resource path is a chain of one or more resource hops.

3.6
RESTful API description language
language designed to provide a structured description of a RESTful Web API that is useful both to humans and for automated machine processing

3.7
simple object access protocol
SOAP
messaging protocol specification for exchanging structured information in the implementation of Web services in computer networks

3.8
WebSocket
protocol which enables two-way communication between a client running untrusted code in a controlled environment to a remote host that has opted-in to communications from that code

3.9
Webhook
user-defined HTTP call-backs triggered by an event such as pushing code to a repository or a comment being posted to a blog

3.10
XML
eXtensible Markup Language, a data format standard created by W3C

4 Design principles

4.1 General

This clause covers absolute principles for developing a Web API. It covers both standalone design principles that shall be considered upfront when developing an API in financial services and principles that apply to specific types of API and are discussed in more detail in Clause 8.

4.2 Standards compatibility

The WAPI advocates the use of ISO data standards wherever possible and practical as a best practice in order to facilitate interoperability.

4.3 Extensibility

Where possible, design decisions for the API ecosystem should be designed to be as extensible as possible. Data standards may change, but the API does not. This is to ensure that their use is able to adapt to future use cases or scenarios.

4.4 Non-repudiation

Non-repudiation is important for the validity of data of exchange in Web APIs and will enforce the confidence in the data to be exchanged. Digital signatures could be used in the context of an API.

4.5 Web resource unique identifiers (ID fields)

A Web resource should have a unique identifier (e.g. a primary key) that can be used to identify the resource. These unique identifiers are used to construct URLs/URIs to identify and address specific resources.

4.6 Idempotency

Idempotency shall be considered upfront for a Web API. This is covered in Clause 8.

4.7 States

There is a need to consider upfront what state will be required.

5 Related technology

5.1 General

This clause talks through related technology and provides some context on its use on APIs in financial services. It does not include terms and definitions, as these are defined in Clause 3, and does not provide specific guidance on how to approach the technology, as this is covered in Clause 8.

5.2 Representational state transfer (REST) and simple object access protocol (SOAP)

5.2.1 General

REST and SOAP provide ways to access Web services, and there are considerations for using both.

5.2.2 REST

REST or RESTful Web services provide the way to transfer the representational state of a resource and to specify the action to be processed on this resource (see Clause 3). REST is the most popular to build a Web service or define Web APIs due to its simplicity and compatibility with existing Internet standards such as HTTP.

APIs typically use REST as it is more efficient (can use smaller message formats) and requires less extensive processing. In most financial scenarios using RESTful APIs, the end user sends a message to the server, which replies shortly after. For example, placing an order in a trade system or requesting the balance on an account.

REST architecture can be exploited to implement these request-response communications. The REST architecture style is described by six constraints[1)]:

— Uniform interface: the uniform interface constraint defines the interface between clients and servers. It simplifies and decouples the architecture, which enables each part to evolve independently.

— Stateless: one client can send multiple requests to the server; however, each of them shall be independent, that is, every request shall contain all the necessary information so that the server can understand it and process it accordingly. In this case, the server shall not hold any information about the client state. Any information status shall stay on client – such as sessions.

— Cacheable: because many clients access the same server, and often request the same resources, it is necessary for these responses to be cached, avoiding unnecessary processing and significantly increasing performance.

— Client-server: the uniform interface separates clients from servers. This separation of concerns means that, for example, clients are not concerned with data storage, which remains internal to each server, so that the portability of client code is improved. Servers are not concerned with the user interface or user state, so that servers can be simpler and more scalable. Servers and clients may also be replaced and developed independently, as long as the interface is not altered.

— Layered system: a client cannot ordinarily tell whether they are connected directly to the end server, or to an intermediary along the way. Intermediary servers may improve system scalability by enabling load-balancing and by providing shared caches. Layers may also enforce security policies.

— Code-on-demand (optional): this condition allows the customer to run some code on demand, that is, extend part of server logic to the client, either through an applet or scripts. Thus, different customers may behave in specific ways even when using exactly the same services provided by the server. As this item is not part of the architecture itself, it is considered optional. It can be used when performing some of the client-side service which is faster or more efficient.

5.2.3 SOAP

SOAP is a messaging protocol specification for exchanging structured information in the implementation of Web services in computer networks. Its purpose is to induce extensibility, neutrality and independence. It uses XML information set for its message format, and relies on application layer protocols, most often Hypertext Transfer Protocol (HTTP) or Simple Mail Transfer Protocol (SMTP), for message negotiation and transmission.

SOAP allows processes running on disparate operating systems (such as Windows and Linux) to communicate using XML. Since Web protocols like HTTP are installed and running on all operating systems, SOAP allows clients to invoke Web services and receive responses independent of language and platforms.

1) R.T. Fielding, *Architectural Styles and the Design of Network-based Software Architectures*. University of California, Irvine, 2000.

5.3 WebSocket and Webhook

5.3.1 General

WebSocket and Webhook protocols exist for scenarios that REST cannot cater for, such as supporting unsolicited responses, specifically:

— client-to-client architectures (Webhook may be used);

— client-to-server architectures (WebSocket may be used);

— any kind of publish/subscribe implementation.

5.3.2 WebSocket

As detailed in RFC6455[2)], the WebSocket protocol enables two-way communication between a client running untrusted code in a controlled environment to a remote host that has opted in to communications from that code. The security model used for this is the origin-based security model commonly used by Web browsers.

The goal of this technology is to provide a mechanism for browser-based applications that need two-way communication with servers that does not rely on opening multiple HTTP connections (e.g. using XMLHttpRequest or <iframe>s and long polling). In the case of massive data transporting or events driven by the server, WebSocket performs better than HTTP, for example market data distribution and market announcement publishing.

Features:

— The WebSocket protocol (IETF RFC 6455) is supported by most browsers, but a client may be any software agent.

— A session is initiated by an HTTP request for protocol upgrade. Transport Layer Security (TLS) handshake is supported for authentication, and cipher suites may be negotiated for privacy and non-repudiation.

— After initial handshake, a session becomes a bidirectional, asynchronous messaging protocol. There is no need for a polling mechanism as either side may push unsolicited messages.

— Two subprotocols are defined: text messages, such as JavaScript object notation (JSON) or XML, and binary messages, such as simple binary encoding. More than one subprotocol may be supported by a peer, if desired. Other subprotocols may be registered with IANA, for example ISO 20022 messages.

— WebSocket is layered over TCP. It inherits its reliability and flow control features. However, WebSocket frames messages over a TCP stream. Therefore, applications need not be concerned with framing.

— No session protocol headers are imposed on application messages. Messages are self-contained and therefore can be serialized or forwarded as a unit (in contrast to HTTP where semantic information is dispersed over payload, URI and headers).

— The session protocol provides no mechanism to correlate requests with responses because it is an event-driven protocol rather than a request/response protocol. Correlation of events is performed at the application layer with transaction IDs and the like.

5.3.3 Webhook

Webhook are sometimes referred to as reverse APIs. Webhook are user-defined HTTP call-backs. They are usually triggered by some event, such as pushing code to a repository or a comment being posted to a blog. When that event occurs, the source site makes an HTTP request to the URL configured for the

2) https://tools.ietf.org/html/rfc6455

Webhook. Users can configure them to cause events on one site to invoke behaviour on another. The action taken may be anything. Common uses are to trigger builds with continuous integration systems or to notify bug-tracking systems. Since they use HTTP, they can be integrated into Web services without adding new infrastructure.

5.4 HTTPS

HTTP is widely used for distributed, collaborative and information systems. Hypertext is structured text that uses logical links (hyperlinks) between nodes containing text. HTTP is the most used protocol to exchange or transfer hypertext.

A later version, the successor, was standardized in 2015, and is now supported by major Web servers and browsers over TLS using the Application-Layer Protocol Negotiation extension where TLS 1.2 or newer is required. In this specification, HTTP/1.1[RFC7230-RFC7235] and HTTP/2.0 are recommended. Although HTTP 1.1 used to be the mainstream of the market, due to the advantages of HTTP 2.0, most browsers have begun to support HTTP 2.0. It is believed that HTTP 2.0 will become popular in the next few years.

HTTPS (HTTP over SSL or HTTP Secure) is the use of Secure Socket Layer (SSL) or TLS as a sublayer under regular HTTP application layering. HTTPS encrypts and decrypts user page requests as well as the pages that are returned by the Web server. The use of HTTPS protects against eavesdropping and man-in-the-middle attacks.

5.5 JSON and XML

5.5.1 General

In Web communication scenarios, the two most used structured data formats for data exchanging are JSON [RFC8259][3] and XML[4].

5.5.2 JSON

JSON is a lightweight, text-based data format widely used in Web-based communication, including as a replacement for XML.

Compared with XML format, JSON has several advantages:

— less verbose than XML, as XML necessitates opening and closing tags, and JSON has concise syntax like using name/value pairs and separated with "{" and "}";

— allows for direct mapping onto the corresponding data structures in the host language, corresponding directly to the object of JavaScript.

JSON is simple, since it supports a variety of server-side languages and native new data. JSON format can be directly used for server-side code, which greatly simplifies the server-side and client-side code development, reducing the consumption of network bandwidth and making it easy to maintain.

5.5.3 XML

XML is a data format standard created by W3C. There are several advantages that XML has over JSON:

— can put metadata into the tags in the form of attributes;

— most browsers render XML in a highly readable and organized way;

— has the concept of schema, supporting strong typing and user-defined types and allowing validation;

3) https://tools.ietf.org/html/rfc8259

4) https://www.w3.org/standards/xml/core

— supports comments.

In summary, JSON is a lightweight solution, easier for a programmer to use; XML is a heavier solution and, while more functional, is restrictive. Each solution applies to a specific financial Web service.

5.6 Content negotiation

Content negotiation refers to the mechanism in which a client and server negotiate the style of content that is returned from the server. The client can request a certain style of document using the following HTTP headers: Accept, Accept-Language, Accept-Charset. These refer to the format of the document, language of the document and the character set of the document, respectively.

On receipt of the HTTP request, a server shall look at the Accept header to determine whether or not it is acceptable. If it is not acceptable then the server shall return an HTTP 406 Not Acceptable status code. If the request is acceptable, the server shall attach the correct MIME type for the request in the Content-Type response header. The server may choose to respect the language and character set preferences of the browser to also format the response. If the server chooses to obey the Accept-Language and Accept-Charset headers, the response headers used should be Content-Language for the Accept-Language header and the Content-Type should be appended with the character set information.

5.7 RESTful API description languages

RESTful API description languages are formal languages designed to provide a structured description of a RESTful Web API that is useful both to a human and for automated machine processing. The structured description might be used to generate documentation for human programmers; such documentation may be easier to read than free-form documentation, since all documentation generated follows the same rules and formatting conventions. Additionally, the description language is usually precise enough to allow automated generation of various software artifacts, like libraries, to access the API from various programming languages, which takes the burden of manually creating them off the programmers.

The RESTful API description language is usually neutral, language-agnostic and industry-agnostic. It does not define API itself, but is useful for designers, programmers and users of an API ecosystem, especially in building up large-scale APIs. In essence, RESTful API description language is a software engineering methodology but a computer architectural or communication technology. The community around RESTful API description languages is active and the landscape is still changing. Up to now, the most active projects in this area have been OpenAPI Specification, RAML and API Blueprint.

— OpenAPI Specification: originally known as the Swagger Specification, this is a specification for machine-readable interface files for describing, producing, consuming and visualizing RESTful Web services. It became an open source collaborative project of the Linux Foundation in 2016. The latest version is 3.0. https://swagger.io/specification/

— RAML: a YAML-based language for describing RESTful APIs. It provides all the information necessary to describe RESTful or practically RESTful APIs. Although designed with RESTful APIs in mind, RAML is capable of describing APIs that do not obey all the constraints of REST (hence the description "practically RESTful"). It encourages reuse, enables discovery and pattern-sharing and aims for merit-based emergence of best practices. https://raml.org/

— API Blueprint: a documentation-oriented Web API description language. The API Blueprint is essentially a set of semantic assumptions laid on top of the Markdown syntax used to describe a Web API. https://apiblueprint.org/

6 Naming conventions

Table 1 lists the applicable character case conventions that WAPIs should follow.

Table 1 — Applicable character case conventions that WAPIs should follow

API element	Rules		Example
HTTP headers	Although RFC 2616 specifies that HTTP headers are case insensitive, it should be best practice to adopt, within an API specification, the same character case for all header definitions [RFC 6648] Custom proprietary "X-"type headers should be avoided	Train-case is the most widely used formalism. Words are capitalised and separated with hyphens (-)	Accept-Charset
		Acronyms should be capitalised	WWW-Authenticate
Query parameters	The most used formalisms are	snake_case: Words are in lower case and separated with underscores (_)	currency_code
		lowerCamelCase: spaces and punctuation are removed and the first letter of each word, except the first one, is capitalised	currencyCode
Resource_path	See below		
Request body	The most used formalisms are	snake_case: Words are in lower case and separated with underscores (_)	currency_code
		lowerCamelCase: spaces and punctuation are removed and the first letter of each word, except the first one, is capitalised	currencyCode
Data types	Above are variables, for data types UpperCamelCase used to define the data type e.g. "country": { "title": "Ctry, Country", "description": "Country of the address.", "$ref": "#/definitions/CountryCode"		CountryCode

7 Resource path

7.1 General

The resource path aims to specify the resources which are relevant for a given API request.

7.2 Resource hops

This resource path is a chain of one or more resource hops. Each resource hop is built on the following components:

— a resource type is mandatory for all resource hops;

— a resource identifier is mandatory except for the last resource hop (see below).