A

WIPO

المنظمة العالمية للملكية الفكرية

CWS/7/4

الأصل: بالإنكليزية

التاريخ: 13 يونيو 2019

اللجنة المعنية بمعايير الويبو

الدورة السابعة

جنيف، من 1 إلى 5 يوليو 2019

اقتراح وضع معيار الويبو بشأن واجهات برمجة التطبيقات على الويب

وثيقة من إعداد المكتب الدولي

مقدمة

.1 مراعاة لتزايد عدد مكاتب الملكية الفكرية التي تستخدم خدمات الويب لدعم عملائها، اقترح المكتب الدولي في الدورة الخامسة للجنة معايير الويبو (لجنة المعايير) المعقودة في عام 2017، إنشاء مهمة جديدة لدعم مكاتب الملكية الفكرية في تطويرها لخدمات الويب (انظر CWS/5/15). وأنشأت لجنة المعايير المهمة رقم 56 وأسندتها إلى فرقة العملXML4IP (XML4IP TF). ويرد وصف المهمة رقم 56 أدناه:

"إعداد توصيات بشأن تبادل البيانات الداعم للتواصل بين الأجهزة، مع التركيز على:

"1" نسق الرسائل، وهيكل البيانات، وقاموس البيانات بنسق JSON أو نسق XML أو كليهما؛

"2" واصطلاحات التسمية لمعرّف الموارد المنتظم (URI)."

.2 وبتنفيذ المهمة رقم 56 وإعداد مشروع المعيار، تأمل فرقة العمل أن تحقق مكاتب الملكية الفكرية الفوائد التالية:

- وجود إرشادات بشأن أفضل ممارسات الصناعة، بغض النظر عن حجم المكتب؛

- ووجود توصية بشأن هيكل البيانات المناسب وقاموس الأعمال الموحد التي من شأنها تيسير التواصل بين الأجهزة أو تطبيقات البرمجيات التي طورتها مكاتب الملكية الفكرية؛

- ووجود توصية بشأن حلول الأمان والمصادقة التي من شأنها أن تساعد مكاتب الملكية الفكرية على اختيار البرمجيات والنهج المتعلقة بالمصادقة للتعامل مع مختلف مستويات الأمان المطلوبة؛

- ووجود اصطلاحات التسمية التي ستوفر نهجًا موحدًا في تحديد موارد البيانات في حين من شأن ترقيم إصدارات الموارد المذكورة أن يُسهّل تبادل بيانات الملكية الفكرية على المستوى الدولي.

.3 وأحاطت لجنة المعايير، في الدورة السادسة المعقودة في 2018، بمعايير مشروع معيار الويبو بشأن واجهات برمجة التطبيقات على الويب التي أعدتها فرقة العمل XML4IP (انظر الوثيقة CWS/6/6CORR). علاوة على ذلك، وافقت لجنة المعايير على مرشحين محتملين كنموذج لواجهات برمجة التطبيقات على الويب: خدمة الويب المستلهمة من ملف البوابة الواحدة (OPD)، وخدمات الويب لتبادل بيانات الوضع القانوني للبراءات (انظر الفقرات 44 إلى 46 من الوثيقة CWS/6/34). ووافقت اللجنة على ضرورة إجراء مزيد من التنقيحات على مشروع المعيار بغية تقديم وثيقة نهائية للنظر فيها في الدورة السابعة للجنة المعايير (انظر الفقرة 48 من الوثيقة CWS/6/34).

.4 وعقدت فرقة العمل XML4IP، منذ الدورة السادسة، في إطار تنفيذ المهمة رقم 56، عدة جولات من المناقشات من خلال مؤتمرات عبر الإنترنت وعبر موقع الويكي التابع لفرقة العمل بغرض تحسين وثيقة المشروع. وتواصلت المناقشات بشأن نطاق المعيار ومحتواه خلال اجتماع فريق العمل XML4IP الذي عقد في سيول في مارس 2019.

.5 وأجرى المكتب الدولي، منذ الدورة السادسة أيضاً، مناقشات داخلية بخصوص صياغة المعيار واعتزم تنفيذه في إطار خدمات الويبو على الويب، عند الاقتضاء.

مراجعة مشروع العمل الخاص بالمعيار.

.6 وأجريت، بناء على المناقشات التي دارت في اجتماع فريق العمل XML4IP الذي عقد في سيول والملاحظات الإضافية المقدمة عبر موقع الويكي التابع لفرقة العمل XML4IP، سلسلة من التعديلات على مشروع معيار الويبو بشأن واجهات برمجة التطبيقات على الويب. وأصبحت الوثيقة المنقحة جاهزة لينظر فيها المشاركون باعتبارها مرفقا أوّلا لهذه الوثيقة.

.7 وفيما يلي ملخص التحديثات التي أجريت منذ أن وُضع الإصدار الأخير في متناول لجنة المعايير في دورتها السادسة:

- خضع المعيار لإعادة الصياغة قصد تحسين التعبير والقواعد النحوية المستخدمة في النص؛

- وصُنّفت قواعد التصميم وفقًا لما إذا كانت تسري على نسق الرد JSON أو XML أو على كليهما. على سبيل المثال، القاعدة [RSG-01] هي قاعدة تسري على واجهات برمجة التطبيقات القائمة على RESTful سواء مع نسق الرد XML أو JSON؛

- وخضعت "قواعد التصميم" لإعادة الصياغة بحيث صارت تستخدم إما مفردة "ينبغي" أو "يجب" دون أن تشتمل على كليهما؛

- وأتيح المرفق الأول على شكل مسودة باستخدام جداول منفصلة لمختلف مستويات الامتثال (’A‘،’AA‘) ومختلف أنساق الرد (XML وJSON)؛

- وأتيح المرفق الثاني في شكل مسودة ويشمل حاليا كلاً من مفردات الأعمال التجارية (الملكية الصناعية) والمفردات التقنية لخدمات الويب القائمة على RESTful وSOAP؛

- وأدرج العقد النموذجي المشار إليه في الفقرة 9 أدناه، على سبيل المثال، كتذييل للمرفق الرابع، لمساعدة القراء على تطوير مواصفات واجهات برمجة التطبيقات الخاصة بهم، والتي تحل محل مواصفات RAML المدرجة أصلاً؛

- وأضيف المرفق الثامن، وهو عبارة عن قائمة مصطلحات التمثيل التي ينبغي استخدامها.

وتجدر الإشارة إلى أنه لا يوجد حتى الآن مثال كنموذج لواجهات برمجة التطبيقات على الويب القائمة على بروتوكول النفاذ البسيط إلى الغرض (SOAP)، على النحو الوارد في التذييل الخامس من مشروع المعيار. وسيناقش فريق العمل XML4IP ملاءمة تقديم مثال بعد الدورة السابعة للجنة المعايير.

.8 وبالتشاور مع مكاتب الملكية الفكرية الخمسة (IP5)[1]، المالكة لنظام ملف البوابة الواحدة OPD، اختيرت خدمة DocList كأفضل مرشح لاستلهام المثال الأول لمواصفات النموذج. وتسمح خدمة DocList الحالية للمستخدمين باسترجاع قائمة محدثة للوثائق المرتبطة برقم طلب معين. وسيحاكي نموذج المثال الجديد هذا الأسلوب، لكن سينفذ أيضا مع استيفاء مستوى الامتثال المعروف بالرمز "AA" في إطار مشروع المعيار الحالي بما يتفق ومقتضيات معيار الويبو ST.96. وللحصول على مزيد من التفاصيل بخصوص مستويات الامتثال المحددة، يرجى الرجوع إلى التذييل الأول لهذه الوثيقة.

.9 ووردت مواصفات عقد الخدمة، التي تشكل تذييل المرفق الرابع لمشروع المعيار والتي أُعدّت لواجهة برمجة التطبيقات لقائمة وثائق WIPO-CASE، لأغراض الإحالة كمرفق ثان (ملف ZIP). وتتكون هذه المواصفات من ملفين: مواصفات واجهة برمجة التطبيقات (API) المكتوبة بنسق YAML ومواصفات عقد الخدمة المكتوبة للوقوف على متطلبات تسيير الأعمال. وهذا مثال على نهج أسلوب "العقد أولا" (انظر الفقرة 12 أدناه).

.10 علاوة على ذلك، يعتزم المكتب الدولي تنفيذ أول مثال النماذج كواجهة برمجة التطبيقات، وهو متاح لمستخدمي نظام WIPO-CASE (نظام النفاذ المركزي إلى البحث والفحص). ولا ترمي واجهة برمجة التطبيقات هذه إلى تعويض خدمة الويب لملف البوابة الواحدة الحالية، ولكن بدلاً من ذلك، تقدم مثالًا على دورة الحياة الكاملة لخدمة الويب التي طُوّرت وفقًا للتوصيات الواردة في مشروع المعيار. وسترى النور خدمة الويب التي جرى تنفيذها في إطار خريطة الطريق الحالية للتطوير بحلول نهاية عام 2019، إلى جانب خدمة ويب جديدة ستقوم أيضًا باسترجاع محتويات تلك الوثائق.

[1] أعضاء مكاتب الملكية الفكرية الخمسة هم إدارة الصين الوطنية للملكية الفكرية والمكتب الأوروبي للبراءات ومكتب كوريا للملكية الفكرية ومكتب اليابان للبراءات ومكتب الولايات المتحدة للبراءات والعلامات التجارية.

مجالات التطوير

11. في هذا الوقت، يطلب فريق عمل XML4IP الإبقاء على المعيار في إطار مشروع العمل ما دام هناك مجالات تستدعي التحسين والتطوير. وعلى هذا النحو، سيُقدم مشروع العمل هذا باللغة الإنجليزية فقط في الوقت الحالي. وتوضح الفقرات التالية الاقتراح المقدم لمراجعة مشروع المعيار لاحقا.

12. ونظرًا لعدم اكتمال مواصفات Open API (OAS) في دعمها لتعريفات مخطط XML(XSD)، سيوفر التذييل الرابع حاليا مجموعة إرشادات تُعنى بتطوير واجهة برمجة تطبيقات الويب، ووضع معايير لتحديد ما إذا كان سيجري صياغة العقد أولاً (المواصفات) أم الرمز أولا أثناء عملية التطوير. ويوصى، على مستوى رفيع، بالنهج التالي:

- في حالة وجود ملفات XSD، على غرار معيار ST.96، فسيكون من الصعب للغاية صياغة المواصفات قبل الرمز وبالتالي لا يوصى بهذه الممارسة؛
- وفي حالة عدم وجود ملفات XSD بالإضافة إلى ضرورة تحديدها من نقطة البداية، فسيكون من الأفضل أن البدء بصياغة بالعقد (المواصفات).

ونتيجةً لذلك، جرى حذف جميع القواعد التي تشير بالتحديد إلى نسق الطلب والرد من مشروع المعيار أو تخفيض حدة اللفظ من "يجب" إلى "ينبغي"، لأغراض الامتثال. وسيتيح هذا لمشروع المعيار دعم هذا النهج.

13. ويطلب من لجنة المعايير تقديم مزيد من الملاحظات قبل إكمال كل من المرفق الأول والمرفق الثاني. ويحبذ، لاسيما تقديم ملاحظات بخصوص النهج المنقح الوارد في المرفق الأول لهذه الوثيقة قصد تقييم مستويات الامتثال مع مشروع المعيار ومدى ملاءمته للاستخدام.

14. وستقدم أمثلة عن نسقي XML وJSON بناءً على معيار الويبو ST.96 لدعم استخدام هذا المعيار لردود واجهة برمجة تطبيقات الويب.

موضوعات عالقة

15. بالإضافة إلى مجالات التطوير للعناصر الموجودة في مشروع المعيار، جرى تحديد المسائل العالقة التالية خلال الاجتماع الذي عقدته فرقة العمل في سيول والتي لا تزال دون حل:

- هل ينبغي الإبقاء على فصل بروتوكول النفاذ البسيط إلى الغرض (SOAP) جزءًا من المعيار؟
- إلى أي درجة ينبغي التشاور مع المطورين المعينين من لدن الدول الأعضاء أو المكتب الدولي أو كليهما؟
- في حالة وفرت مكاتب الملكية الفكرية بيانات الوضع القانوني للبراءة الواجب التي تستخدمها واجهة برمجة التطبيقات، فهل ستقدم مجموعة البيانات كحزمة بيانات أم بواسطة حقوق معينة للملكية الفكرية وما هي الوثيرة التي سيتبعها التحديث؟

- هل ينبغي وضع إرشادات بخصوص إنشاء Sandbox للتجربة وكيف سيكون عمل الأمن الخاص بواجهة برمجة التطبيقات؟

- هل نطاق نماذج الأمن المقدمة في المعيار هو نطاق صارم للغاية؟

- هل تهتم مكاتب الملكية الفكرية بتطوير المزيد من واجهات برمجة التطبيقات المستلهمة من ملف البوابة الواحدة (OPD) المتوافقة مع المعايير؟

مقترح لتعديل المهمة رقم 56

.16 في الوقت الحالي، يُدير فريق العمل XML4IP المهمة رقم 56. ومع ذلك، يقترح فريق العمل إنشاء فريق عمل جديد لإدارة هذه المهمة لأن فريق العمل يتشكل من خبراء في مجال الأعمال التجارية الملمّين بنسق XML ولكن ليس بالضرورة على دراية بتطوير واجهات برمجة التطبيقات (انظر الفقرة 33 من الوثيقة CWS/7/3).

.17 بناءً على موافقة لجنة المعايير إنشاء فرقة العمل الجديدة هذه، يقترح المكتب الدولي بالإضافة إلى ذلك النظر فيما يلي:

(أ) تعديل وصف المهمة رقم 56 ليصبح النص كما يلي:

"إعداد توصيات بشأن تبادل البيانات الداعم للتواصل بين الأجهزة مع التركيز على: "1" تيسير تطوير خدمات الويب التي يتم في نطاقها الحصول على موارد الملكية الفكرية "2" وتوفير قاموس الأعمال وهيكل البيانات المناسب و "3" اصطلاحات التسمية لمعرف المورد المنتظم (URI).

(ب) إنشاء منتدى عبر الإنترنت للسماح بتظافر جهود تعاونية واسعة النطاق بين فرقة العمل الحديثة العهد والمطورين حاليًا، وربما في المستقبل، وتطوير واجهات برمجة التطبيقات للنفاذ إلى موارد الملكية الفكرية.

.18 إن لجنة المعايير مدعوة للقيام بما يلي:

(أ) الإحاطة علما بمحتوى هذه الوثيقة والمرافق؛

(ب) النظر في الاقتراح المتعلق بإنشاء فرقة العمل الجديدة والبت فيه، على النحو المشار إليه في الفقرة 16 أعلاه؛

(ج) النظر في اقتراحات تعديل المهمة رقم 56 وإنشاء منتدى على الإنترنت على النحو المبين في الفقرة 17 أعلاه؛

(د) تشجيع مكاتب الملكية الفكرية على المشاركة في اختبار واجهات برمجة التطبيقات الجديدة لنظام WIPO CASE فور تنفيذها، على النحو المشار إليه في الفقرة 10؛

(ه) طلب بأن تقدم مكاتب الملكية الفكرية ملاحظات بشأن المرفقات المنقحة أو المرفقات الجديدة لمشروع المعيار، على النحو المشار إليه في الفقرات 7 و12 و13، بما في ذلك نطاق المرفقات؛

(و) طلب بأن يصدر المكتب الدولي تعميما يدعو فيه مكاتب الملكية الفكرية ترشيح خبرائها المختصين في تطوير واجهات برمجة تطبيقات الويب للمشاركة في فرقة العمل الجديدة، في حالة الموافقة عليها؛

(ز) طلب بأن تقدم فرقة العمل الجديدة مقترحا نهائيا للمعيار الجديد.

[يلي ذلك المرفقات]

CWS/7/4
ANNEX I
المرفق الأول

WIPO STANDARD ST.XX

RECOMMENDATIONS FOR WEB API ON INTELLECTUAL PROPERTY DATA

Working Draft - version 0.9.0

Editorial Note prepared by the International Bureau

This Working Draft is prepared by the XML4IP Task Force and shared for information at the seventh session of the CWS, only in English. This Draft will be further updated in due course and the final draft will be submitted for consideration by the CWS at its eighth session.

TABLE OF CONTENTS

1. INTRODUCTION 3
2. DEFINITIONS AND TERMINOLOGY 3
3. Notations 4
3.1. General notations 4
3.2. Rule identifiers 4
4. SCOPE 5
5. WEB API DESIGN PRINCIPLES 5
6. RESTFUL WEB API 7
6.1. URI Components 7
6.2. Status Codes 8
6.3. Resource Model 8
6.4. Supporting multiple formats 10
6.5. HTTP Methods 10
6.6. Data Query Patterns 14
6.7. Error Handling 16
6.8. Service Contract 17
6.9. Time-out 18
6.10. State Management 18
6.11. Preference Handling 19
6.12. Translation 19
6.13. Long-Running Operations 19
6.14. Security Model 20
6.15. API Maturity Model 23
7. SOAP WEB API 24
7.1. General Rules 24
7.2. Schemas 25
7.3. Naming and Versioning 25
7.4. Web Service Contract Design 26
7.5. Attaching Policies to WSDL Definitions 26
7.6. SOAP – Web Service Security 26
8. Data Type Formats 27
9. CONFORMANCE 27
10. REFERENCES 28
WIPO Standards 28
Standards and Conventions 28
IP Offices' REST APIs 29

Industry REST APIs and Design Guidelines 29
Others 30
ANNEX I - LIST OF RESTful WEB SERVICE DESIGN RULES AND CONVENTIONS 31
ANNEX II – REST IP Vocabulary 59
ANNEX III - LIST OF SOAP Web API NAMES 60
ANNEX IV – RESTFUL WEB API GUIDELINES AND MODEL SERVICE CONTRACT 60
Appendix 60
ANNEX V - SOAP WEB API MODEL SERVICE CONTRACT 60
ANNEX VI – HIGH LEVEL SECURITY ARCHITECTURE BEST PRACTICES 61
ANNEX VII – HTTP STATUS CODES 61
ANNEX VIII – REPRESENTATION TERMS 64

1. INTRODUCTION

1. This Standard provides recommendations on Application Programming Interface (API) to facilitate the processing and exchange of Intellectual Property (IP) data in harmonized way over the Web.

2. This Standard is intended to:

- ensure consistency by establishing uniform web service design principles;
- improve data interoperability among web service partners;
- encourage reusability through unified design;
- promote data naming flexibility across business units through a clearly defined namespace policy in associated XML resources;
- promote secure information exchange;
- offer appropriate internal business processes as value-added services that can be used by other organizations;
- integrate its internal business processes and dynamically link them with business partners.

2. DEFINITIONS AND TERMINOLOGY

3. For the purpose of this Standard, the expression:

- The term “Hyper Text Transfer Protocol (HTTP)” is intended to refer to the application protocol for distributed, collaborative, and hypermedia information systems. HTTP is the foundation of data communication for the World Wide Web. HTTP functions as a request–response protocol in the service oriented computing model.
- The term “Application Programming Interfaces” (API) means software components that provide a reusable interface between different applications that can easily interact to exchange data.
- The term “Representational State Transfer (REST)” describes a set of architectural principles by which data can be transmitted over a standardized interface, i.e., HTTP. REST does not contain an additional messaging layer and focuses on design rules for creating stateless services.
- The term “Simple Object Access Protocol (SOAP)” means a protocol for sending and receiving messages between applications without confronting interoperability issues. SOAP defines a standard communication protocol (set of rules) specification for XML-based message exchange. SOAP uses different transport protocols, such as HTTP and SMTP. The standard protocol HTTP makes it easier for SOAP model to tunnel across firewalls and proxies without any modifications to the SOAP protocol.
- The term “Web Service” means a method of communication between two applications or electronic machines over the World Wide Web (WWW) and Web Services are of two kinds: REST and SOAP.
- “RESTful Web API” means a set of Web Services based on REST architectural paradigm and typically use JSON or XML to transmit data.
- “SOAP Web API” means a set of SOAP Web Services based on SOAP and mandate the use of XML as the payload format.
- The term “Web Services Description Language (WSDL)" means a W3C Standard that is used with the SOAP protocol to provide a description of a Web Service. This includes the methods a Web Service uses, the parameters it takes and the means of locating Web Services etc.
- The term RAML refers to the RESTful API Modelling Language, a language which allows developers to provide a specification of their API.
- The terms OAS refers to the Open API Specification, a
- The term “Service Contract” (or Web Service Contract) means a document that expresses how the service exposes its capabilities as functions and resources offered as a published API by the service to other software programs; the term “REST API documentation” is interchangeably used for the Service Contract for RESTful Web APIs.
- The term “Service Provider” means a Web Service software exposing a Web Service.
- The term “Service Consumer” means the runtime role assumed by a software program when it accesses and invokes a service. More specifically, when the program sends a message to a service capability expressed in the service contract. Upon receiving the request, the service begins processing and it may or may not return a corresponding response message to the service consumer.
- The term “camelcase” is either the lowerCamelCase (e.g., applicantName), or the UpperCamelCase (e.g., ApplicantName) naming convention.
- The term kebab-case is one of the naming conventions where all are lowercase with hyphens “-“ separating words, for example a-b-c.
- The term “Open Standards” means the standards that are made available to the general public and are developed (or approved) and maintained via a collaborative and consensus driven process. “Open Standards” facilitate interoperability and data exchange among different products of services and are intended for widespread adoption.
- Uniform Resource Identifier (URI) identifies a resource and Uniform Resource Locator (URL) is a subset of the URIs that include a network location.

- The term “Entity Tag (ETag)” means an opaque identifier assigned by a web server to a specific version of a resource found at a URL. If the resource representation at that URL ever changes, a new and different ETag is assigned. ETags can be compared quickly to determine whether two representations of a resource are the same.
- The term “Service Registry” means a network-based directory that contains available services.
- The term “Semantic Versioning” means a versioning scheme where a version is identified by the version number MAJOR.MINOR.PATCH, where:
  - MAJOR version when you make incompatible API changes,
  - MINOR version when you add functionality in a backwards-compatible manner and
  - PATCH version when you make backwards-compatible bug fixes.

4. In terms of conformance in design rules the following keywords should be interpreted, in the same manner as defined in para. 8 of WIPO ST.96[2], that is:

- MUST: an equivalent to “REQUIRED” or “SHALL”, means that the definition is an absolute requirement of the specification;
- MUST NOT: equivalent to “SHALL NOT”, means that the definition is an absolutely prohibited by the specification;
- SHOULD: equivalent to “RECOMMENDED”, means that there may exist valid reasons for ignoring this item, but the implications of doing so need to be fully considered;
- SHOULD NOT: equivalent to “NOT RECOMMENDED”, means that there may exist valid reasons where this behavior may be acceptable or even useful but the implications of doing so need to be carefully considered; and
- MAY: equivalent to “OPTIONAL”, means that this item is truly optional, and is only provided as one option selected from many.

3. NOTATIONS

3.1. General notations

5. The following notations are used throughout this document:

- <>: Indicates a placeholder descriptive term that, in implementation, will be replaced by a specific instance value.
- “ ”: Indicates that the text included in quotes must be used verbatim in implementation.
- { }: Indicates that the items are optional in implementation.
- `Courier font`: Indicates keywords or source code.

3.2. Rule identifiers

6. All design rules are normative. Design rules are identified through a prefix of [XX-nn] or [XXY-nn].

(a) The value “XX” is a prefix to categorize the type of rule as follows:

- WS for SOAP Web API design rules
- RS for RESTful Web API design rules
- CS for both SOAP and RESTful WEB API design rule

(b) The value “Y” is used only for ReSTful design rules and provides further granularity on the type of response that the rule is related to:

- . “G” indicates it is a general rule for both JSON and XML response;
- “J” indicates it is for a JSON response; and
- “X” indicates it is an XML response.

(c) The value “nn” indicates the next available number in the sequence of a specific rule type. The number does not reflect the position of the rule, in particular, for a new rule. A new rule will be placed in the relevant context. For example, the rule identifier [WS-4] identifies the fourth SOAP Web API design rule. The rule [WS-4] can be placed between rules [WS-10] and [WS-11] instead of following [WS-3] if that is the most appropriate location for this rule.

(d) The rule identifier of the deleted rule will be kept while the rule text will be replaced with “Deleted”.

Please refer to the References chapter [2]

4. SCOPE

7. This Standard aims to guide the Intellectual Property Offices (IPOs) and other Organizations that need to manage, store, process, exchange and disseminate IP data using Web APIs. It is intended that by using this Standard, the development of Web APIs can be simplified and accelerated in a harmonized manner and interoperability among Web APIs can be enhanced.

8. This Standard intends to cover the communications between IPOs and their applicants or data users, and between IPOs through connections between devices-to-devices and devices-to-software applications.

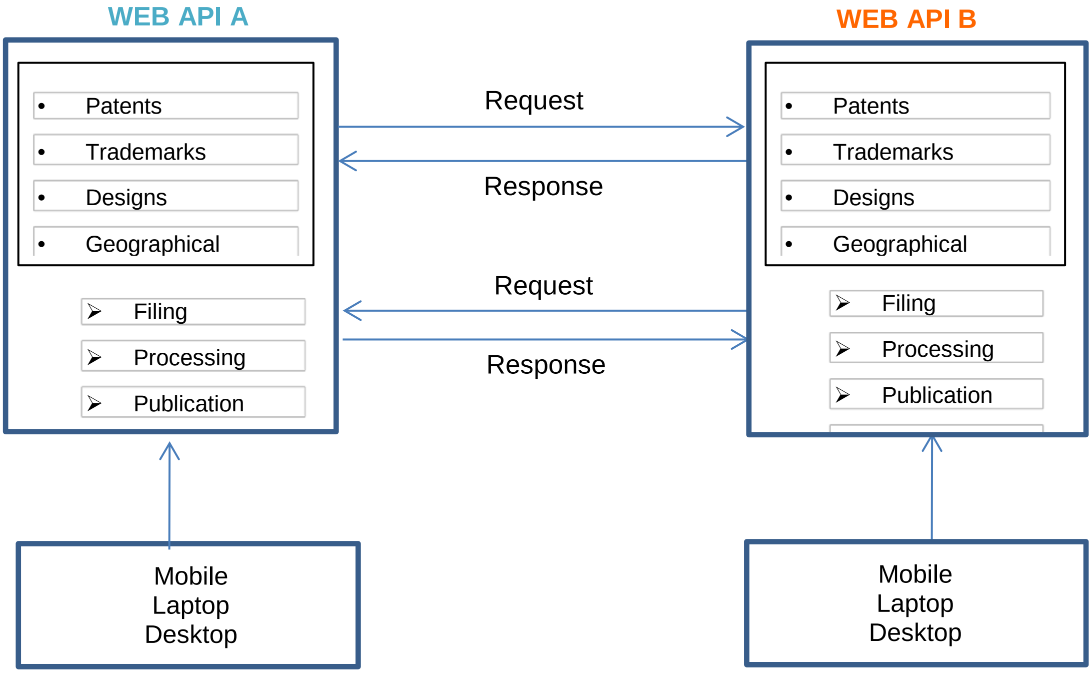


Fig. 1 Scope of the Standard

9. This Standard is to provide a set of design rules and conventions for RESTful Web APIs and SOAP Web APIs; list of IP data resources which will be exchanged or exposed; model API documentation or service contract, which can be used for customization, describing message format, data structure and data dictionary in JSON[3] and/or XML based on WIPO Standard ST.96 and mock-up (reference) APIs to be used by IPOs. This Standard provides guidelines for RESTful Web API in detail and SOAP Web APIs in much less detail on demand.

10. This Standard provides model Service Contracts for SOAP Web APIs using WSDL and, for RESTful Web APIs using the REST API Modeling Language (RAML) and Open API Specification (OAS). A Service Contract also defines or refers to data types for interfaces (see the Section “Data Type Convention” below). This Standard recommends three types of interfaces: REST-XML (XSD), REST-JSON and SOAP-XML (XSD).

11. This Standard excludes the following:

(a) Binding to specific implementation technology stacks and commercial off-the-shelf (COTS) products;

(b) Binding to specific architectural designs (for example, Service Oriented Architecture (SOA) or Microservice Architecture);

(c) Binding to specific algorithms such as algorithms for the calculation of ETag, i.e., calculation of a unique identifier for a specific version of a resource (for example, used for caching).

5. WEB API DESIGN PRINCIPLES

12. Both RESTful Web APIs and SOAP Web APIs have proven their ability to meet the demands of big organizations as well as to service the small-embedded applications in production. When choosing between RESTful and SOAP, the following aspects can be considered:

The WIPO JSON standard is currently under discussion but will be based on WIPO Standard ST.96[3]

- Security, e.g., SOAP has WS-Security while REST does not specify any security patterns;
- ACID Transaction, .e.g., SOAP has WS-AT specification while REST does not have a relevant specification;
- Architectural style, e.g., Microservices and Serverless Architecture Style use REST while SOA uses SOAP web services;
- Flexibility;
- Bandwidth constraints;
- Guaranteed delivery, e.g. SOAP offers WS-RM while REST does not have a relevant specification.

13. The following service-oriented design principles should be respected when a Web API is designed:

(a) Standardized Service Contract: Standardizing the service contracts is the most important design principle because the contracts allow governance and a consistent service design. A service contract should be easy to implement and understand. A service contract consists of metadata that describes how the service provider and consumer will interact. Metadata also describes the conditions under which those parties are entitled to engage in an interaction. It is recommended that service contracts include:

- Functional requirements: what functionality the Service provides and what data it will return, or typically a combination of the two;
- Non-functional requirements: information about the responsibility of the providers for providing their functionality and/or data, as well as the expected responsibilities of the consumers of that information and what they will need to provide in return. For example, a consumer's availability, security, and other quality of service considerations.

(b) Service Loose Coupling: Clients and services should evolve independently. Applying this design principle requires:

- Service versioning – Consumers bound to a Web API version should not take the risk of unexpected disruptions due to incompatible API changes.
- The service contract should be independent of the technology details.

(c) Service Abstraction– The service implementation details should be hidden. The API Design should be independent of the strategies supported by a server. For example, for the REST Web Service, the API resource model should be decoupled from the entity model in the persistence layer.

(d) Service Statelessness – Services should be scalable.

(e) Service Reusability – A well-designed API should provide reusable services with generic contracts. In this regard, this Standard provides a model service contract.

(f) Service Autonomy – The Service functional boundaries should be well defined.

(g) Service Discoverability –Services should be effectively discovered and interpreted.

(h) Service Composability Services can be used to compose other services.

(i) Using Standards as a Foundation – The API Should follow industry standards (such as IETF, ISO, and OASIS) wherever applicable, naturally favoring them over locally optimized solutions.

(j) Pick-and-choose Principle – It is not required to implement all the API design rules. The design rules should be chosen based on the implementation of each concrete case.

14. In addition, the following principles should be respected especially with regard to the RESTful Web APIs:

(a) Cacheable: responses explicitly indicate their cacheability;

(b) Resource identification in requests: individual resources are identified in requests; for example using URIs in Web-based REST systems. The resources themselves are conceptually separate from the representations that are returned to the client.

(c) Hypermedia as the engine of application state (HATEOAS) - having accessed an initial URI for the REST application—analogous to an individual accessing the home page of a website—a REST client should then be able to use server-provided links dynamically to discover all the available actions and resources it needs.

(d) Resource manipulation through representations - when a client holds a representation of a resource, including any metadata attached, it has enough information to modify or delete the resource.

(e) Self-descriptive messages - each message includes enough metadata to describe how to process the message content.

(f) Web API should follow HTTP semantics such as methods, errors etc.

(g) Available to the public - design with the objective that the API will eventually be accessible from the public internet, even if there are no plans to do so at the moment.

(h) Common authentication - use a common authentication and authorization pattern, preferably based on existing security components, in order to avoid creating a bespoke solution for each API.

(i) Least Privilege - access and authorization should be assigned to API consumers based on the minimal amount of access they need to carry out the functions required.

(j) Maximize Entropy - the randomness of security credentials should be maximized by using API Keys rather than username and passwords for API authorization, as API Keys provide an attack surface that is more challenging for potential attackers.

(k) Performance versus security - balance performance with security with reference to key life times and encryption / decryption overheads.

6. RESTFUL WEB API

15. A RESTful Web API allows requesting systems to access and manipulate textual representations of Web resources using a uniform and predefined set of stateless operations.

6.1. URI Components

16. RESTful Web API s use URIs to address resources. According to RFC 3986, an URI syntax should be defined as follows:

URI = <scheme> "://" <authority> "/" <path> {"?" query}

authority = {userinfo@}host{:port}

```
 For example, https://wipo.int/api/v1/patents?sort=id&offset=10
              ______/______/___________/_________________/
                |      |         |          |
            scheme authority   path   query parameters
```

17. The forward slash “/” character is used in the path of the URI to indicate a hierarchical relationship between resources but the path must not end with a forward slash as it does not provide any semantic value and may cause confusion.

[RSG-01] The forward slash character “/” MUST be used in the path of the URI to indicate a hierarchical relationship between resources but the path MUST NOT end with a forward slash.

18. URIs are case sensitive except for the scheme and host parts. For example, although `https://wipo.int/api/my-resources/uniqueId` and `https://wipo.INT/api/my-resources/uniqueId` are the same, `https://wipo.int/api/my-resources/uniqueid` is not. For the resource names, the kebab-case and the lowerCamelCase conventions provide good readability and maps the resource names to the entities in the programming languages with simple transformation. For the query parameters, the lowerCamelCase should be used. For example, `https://wipo.int/api/v1/inventors?firstName=John`. Resource names and query parameter are all case sensitive. Note, that resource names and query parameter names may be abbreviated.

19. A RESTful Web API may have arguments:

- In the query parameter; for example, /inventors?id=1;
- In the matrix parameter; for example, /inventors;id=1;
- In the URI path segment parameter, for example, /inventors/1; and
- In the request payload such as part of a JSON body.

20. Except of the aforementioned argument types, which are part of the URI, an argument can also be part of the request payload.

[RSG-02] Resources name MUST be consistent in their naming pattern.

[RSG-03] Resource names in the request SHOULD kebab-case naming conventions they MAY be abbreviated.

[RSG-04] Query parameters MUST be consistent in their naming pattern

[RSG-05] Query parameters SHOULD use the lowerCamelCase convention and they MAY be abbreviated.

21. A Web API endpoint must comply with `IETF RFC 3986` and should avoid potential collisions with page URLs for the website hosted on the root domain. A Web API needs to have one exact entry point to consolidate all requests. In general, there are two patterns of defining endpoints:

- As the first path segment of the URI, for example: https://wipo.int/api/v1/; and
- As subdomain, for example: https://api.wipo.int/v1/

[RSG-06] The URL pattern for a Web API MUST contain the word "`api`" in the URI.

22. Matrix parameters are an indication that the API is complex with multiple levels of resources and sub-resources. This goes against the service-oriented design principles, previously defined. Moreover, matrix parameters are not standard as they apply to a particular path element while query parameters apply to the request as a whole. An example of matrix parameters is the following: `https://api.wipo.int/v1/path;param1=value1;param2=value2`.

[RSG-07] Matrix parameters MUST NOT be used.

6.2. Status Codes

23. A Web API must consistently apply HTTP status codes as described in IETF RFCs. HTTP status codes should be used among the ones listed in the standard HTTP status codes (RFC 7807) reproduced in Annex VII.

[RSG-08] A Web API MUST consistently apply HTTP status codes as described in IETF RFCs

[RSG-09] The recommended codes in Annex VI SHOULD be used by a Web API to classify the error.

6.2.1 Pick-and-choose Principle

24. A Service Contract should be tolerant to unexpected parameters (in the request, using query parameters) but raise an error in case of malformed values on expected parameters.

[RSG-10] If the API detects invalid input values, it MUST return the HTTP status code "`400 Bad Request`". The error payload MUST indicate the erroneous value.

[RSG-11] If the API detects syntactically correct argument names (in the request or query parameters) that are not expected, it SHOULD ignore them.

[RSG-12] If the API detects valid values that require features to not be implemented, it MUST return the HTTP status code "`501 Not Implemented`". The error payload MUST indicate the unhandled value.

6.3. Resource Model

25. An IP data model should be divided into bounded contexts following a domain-driven design approach. Each bounded context must be mapped to a resource. According to the design principles, a Web API resource model should be decoupled from the data model. A Web API should be modeled as a resource hierarchy to leverage the hierarchical nature of the URI to imply structure (association or composition or aggregation), where each node is either a simple (single) resource or a collection of resources.

26. In this hierarchical resource model, the nodes in the root are called 'top-level nodes' and all of the nested resources are called 'sub-resources'. Sub-resources should be used only to imply compositions, i.e., resources that cannot be top-level resources, otherwise there would be multiple way of retrieving the same entities. Such sub-resources, implying association, are called sub-collections. The other hierarchical structures, i.e., association and aggregation, should be avoided to avoid complex APIs and duplicate functionality.

27. The endpoint always determines the type of the response. For example, the endpoint `https://wipo.int/api/v1/patents` returns always responses regarding patent resources. The endpoint `https://wipo.int/api/v1/patents/1/inventor` returns always responses regarding inventor resources. However the endpoint `https://wipo.int/api/v1/inventors` is not allowed because the inventor resource should be cannot be standalone.

28. Only top-level resources, i.e. with a maximum of one level should be used, otherwise these APIs will be very complex to implement. For example, `https://wipo.int/api/v1/patents?inventorId=12345` should be used instead of `https://wipo.int/api/v1/inventors/12345/patents`.

[RSG-13] A Web API SHOULD only use top-level resources. If there are sub-resources, they should be collections and imply an association. An entity should be accessible as either top-level resource or sub-resource but not using both ways.

[RSG-14] If a resource can be stand-alone it MUST be a top-level resource, or otherwise a sub-resource.

[RSG-15] Query parameters MUST be used instead of URL paths to retrieve nested resources.

29. A Web API should support projection. If only specific attributes from the retrieved data are required, a projection query parameter must be used instead of URL paths. The query parameter should be formed as follows: “`fields=”<comma-separated list of attribute names>`. A projection query parameter is easier to implement and can retrieve multiple attributes. For example, with a JSON response:

```
GET https://wipo.int/api/v1/inventors/id12345?fields=firstName,lastName
200 OK
{ …
  “firstName”: “My first name”,
  "lastName": "My last name"

}
```

[RSG-16] A query parameter SHOULD be used instead of URL paths in case that a Web API supports projection following the format: “`fields=”<comma-separated list of attribute names>`.

30. There are types[4] of Web APIs: the CRUD (Create, Read, Update, and Delete) Web API and the Intent Web API. CRUD Web APIs model changes to a resource, i.e., create/read/update/delete operations. Intent Web APIs by contrast model business operations, e.g., renew/register/publish. CRUD operations should use nouns and Intent Web APIs should use verbs for the resource names. CRUD Web APIs are the most common but both can be combined for example, the service consumer could use an Intent Web API modeling business operation, which would orchestrate the execution of one or more CRUD Web APIs service operations. Using CRUD Web API, the service caller has to orchestrate the business logic but with Intent Web APIs it is the service provider who orchestrates the business logic. CRUD Web APIs are not atomic when compared with Intent Web APIs[5].

– For example, the owner of the IP right wants to locate their patent and renew it. This is a business operation so a CRUD Web API would model this operation in a non-atomic process, requiring two actions such as:

```
GET https://wipo.int/api/v1/patents/id12345
{
  …
}
POST https://wipo.int/api/v1/renewals
{
  …
}
```

– The previous example could also be modeled with an atomic service call using an Intent Web API such as:

```
POST https://wipo.int/api/v1/findAndRenew/id12345
```

31. The type of Web API should then place constraints on how the resources are named to provide an indication on which is being used. Note, that resource names that are localized due to business requirements may be in other languages.

[RSG-17] Resource names SHOULD be nouns for CRUD Web APIs and verbs for Intent Web APIs.

[RSG-18] If resource name is a noun it SHOULD always use the plural form. Irregular noun forms SHOULD NOT be used. For example, /persons should be used instead of /people.

[RSG-19] Resource names, segment and query parameters MUST be composed of words in the English language, using the primary English spellings provided in the Oxford English Dictionary. Resource names that are localized due to business requirements MAY be in other languages.

Alternatively we could classify APIs according to their archetype. See for instance: “REST API Design Rulebook: Designing Consistent RESTful Web Service Interfaces”[4]

An Intent API also enables the application of the Command Query Responsibility Segregation (CQRS) pattern. CQRS is a pattern, where you can use a different model to update information than the [5] model you use to read information. The rationale is that for many problems, particularly in more complicated domains, having the same conceptual model for commands and queries leads to a more complex model that is not beneficial.

6.4. Supporting multiple formats

32. Different service consumers may have differing requirements for the data format of the service responses. The media type of the data should be decoupled from the data itself, allowing the service to support a range of media types. Therefore, a Web API must support content type negotiation using the request HTTP header `Accept` and the response HTTP header `Content-Type` as required by IETF RFC 7231. Additionally, a Web API may support other ways of content type negotiation such as query parameter (for example `?format`) or URL suffix (for example `.json`).

[RSG-20] A Web API SHOULD use for content type negotiation the request HTTP header `Accept` and the response HTTP header `Content-Type`.

33. APIs must support XML and JSON requests and responses. For XML, responses must be compliant with WIPO Standard using XML such as ST.96[6]. A consistent mapping between these two formats should be used. This Standard recommends the BadgerFish convention due to its simplicity until the JSON specification is provided.

[RSG-21] A Web API MUST support content type negotiation following IETF RFC 7231.

[RSG-22] JSON format MUST be assumed when no specific content type is requested.

[RSG-23] A Web API SHOULD return the status code "`406 Not Acceptable`" if a requested format is not supported.

[RSG-24] A Web API SHOULD reject requests containing unexpected or missing content type headers with the HTTP status code "`406 Not Acceptable`" or "`415 Unsupported Media Type`".

[RSX-25] The requests and responses (naming convention, message format, data structure, and data dictionary) SHOULD refer to WIPO Standard ST.96.

[RSJ-26] JSON object property names SHOULD be provided in lowerCamelCase, e.g., applicantName.

[RSX-27] XML component names SHOULD be provided in UpperCamelCase.

[RSG-28] A Web API MUST support at least XML or JSON.

6.5. HTTP Methods

34. HTTP Methods (or HTTP Verbs) are a type of function provided by a uniform contract to process resource identifiers and data. HTTP Methods must be used as they were intended to according the standardized semantics as specified in IETF RFC 7231 and 5789, namely:

- `GET` – retrieve data
- `HEAD` – like `GET` but without a response payload
- `POST` – submit new data
- `PUT` – update
- `PATCH` – partial update
- `DELETE` – delete data
- `TRACE` – echo
- `OPTIONS` – query verbs that the server supports for a given URL

35. The uniform contract establishes a set of methods to be used by services within a given collection or inventory. HTTP Methods tunneling may be useful when HTTP Headers are rejected by some firewalls.

36. HTTP Methods may follow the 'pick-and-choose' principle, which states that only the functionality needed by the target usage scenario should be implemented. Some proxies support only `POST` and `GET` methods. To overcome these limitations, a Web API may use a POST method with a custom HTTP header "tunneling" the real HTTP method. HTTP Methods may also follow the pick-and-choose principle, which states that only the functionality needed by the target usage scenario should be implemented.

[RSG-29] HTTP Methods MUST be restricted to the HTTP standard methods `POST`, `GET`, `PUT`, `DELETE`, `OPTIONS`, `PATCH`, `TRACE` and `HEAD`, as specified in IETF RFC 7231 and 5789.

[RSG-30] HTTP Methods MAY follow the pick-and-choose principle, which states that only the functionality needed by the target usage scenario should be implemented.

A JSON specification is currently under discussion at WIPO and will be available as a Standard in the future. [6]

[RSG-31] Some proxies support only `POST` and `GET` methods. To overcome these limitations, a Web API MAY use a POST method with a custom HTTP header “tunneling” the real HTTP method. The custom HTTP header `X-HTTP-Method` SHOULD be used.

[RSG-32] If a HTTP Method is not supported, the HTTP status code “`405 Method Not Allowed`” SHOULD be returned.

37. In some use cases, multiple operations should be supported at once.

[RSG-33] A Web API SHOULD support batching operations (aka bulk operations) in place of multiple individual requests to achieve latency reduction. The same semantics should be used for HTTP Methods and HTTP status codes. The response payload SHOULD contain information about all batching operations. If multiple errors occur, the error payload SHOULD contain information about all the occurrences (in the details attribute). All bulk operations SHOULD be executed in an atomic operation.

GET

38. According to IETF RFC 2616, the HTTP protocol does not place any a prior limit on the length of a URI. On the other hand, servers should be cautious about depending on URI lengths above 255 bytes, because some older client or proxy implementations may not properly support these lengths. In the case where this limit is exceeded, it is recommended that named queries are used. Alternatively, a set of rules which determine how to convert between and GET and a POST must be specified. According to the IETF RFC 2616, a GET request must be idempotent, in that the response will be the same no matter how many times the request is run.

[RSG-34] For an end point which fetches a single resource, if a resource is not found, the method `GET` MUST return the status code “`404 Not Found`”. Endpoints which return lists of resources will simply return an empty list.

[RSG-35] If a resource is retrieved successfully, the `GET method` MUST return `200 OK`.

[RSG-36] A `GET` request MUST be idempotent.

[RSG-37] When the URI length exceeds the 255 bytes, then the `POST` method SHOULD be used instead of `GET` due to `GET` limitations, or else create named queries if possible.

HEAD

39. When a client needs to learn information about an operation, they can use `HEAD`. `HEAD` gets the HTTP header you would get if you made a `GET` request, but without the body. This lets the client determine caching information, what content-type would be returned, what status code would be returned. A `HEAD` request MUST be idempotent according to the IETF RFC 2616.

[RSG-38] A `HEAD` request MUST be idempotent.

[RSG-39] Some proxies support only `POST` and `GET` methods. A Web API SHOULD support a custom HTTP request header to override the HTTP Method in order to overcome these limitations.

POST

40. When a client needs to create a resource, they can use `POST`. For example,

```
POST https://wipo.int/v1/patents
{ "title": "Patent Title" }
Response:
201 Created
Location: https://wipo.int/v1/patents/id12345
{  "id": id12345, "title": "Patent Title" }
```

[RSG-40] A `POST` request MUST NOT be idempotent according to the IETF RFC 2616.

[RSG-41] If the resource creation was successful, the HTTP header `Location` SHOULD contain a URI (absolute or relative) pointing to a created resource.

[RSG-42] If the resource creation was successful, the response SHOULD contain the status code “`201 Created`”.

[RSG-43] If the resource creation was successful, the response payload SHOULD by default contain the body of the created resource, to allow the client to use it without making an additional HTTP call.

PUT

41. When a client needs to replace an existing resource entirely, they can use `PUT`. Idempotent characteristics of `PUT` should be taken into account. A `PUT` request has an update semantic (as specified in IETF RFC 7231) and an update or insert semantic.

[RSG-44] A `PUT` request MUST be idempotent .

[RSG-45] If a resource is not found, `PUT` MUST return the status code “`404 Not Found`”.

[RSG-46] If a resource is updated successfully, `PUT` MUST return the status code “`200 OK`” if the updated resource is returned or a “`204 No Content`” if it is not returned.

PATCH

42. When a client requires a partial update, they can use `PATCH`. Idempotent characteristics of `PATCH` should be taken into account. For example:

```
PATCH https://wipo.int/v1/patents/id12345

If-Match:456

Content-Type: application/merge-patch+json

{ "Title": "Patent Title" }
```

43. PATCH must not be idempotent according to IETF RFC 2616. . In order to make it idempotent, the API may follow the IETF RFC 5789 suggestion of using optimistic locking.

[RSG-47] A `PATCH` request MUST NOT be idempotent.

[RSG-48] If a Web API implements partial updates, idempotent characteristics of `PATCH` SHOULD be taken into account. In order to make it idempotent the API MAY follow the IETF RFC 5789 suggestion of using optimistic locking.

[RSG-49] If a resource is not found `PATCH` MUST return the status code “`404 Not Found`”.

[RSJ-50] If a Web API implements partial updates using `PATCH`, it MUST use the JSON Merge Patch format to describe the partial change set, as described in IETF RFC 7386 (by using the content type `application/merge-patch+json`.

DELETE

44. When a client needs to delete a resource, they can use `DELETE`. A `DELETE` request must not be idempotent according to the IETF RFC 2616

[RSG-51] A `DELETE` request MUST NOT be idempotent.

[RSG-52] If a resource is not found, `DELETE` MUST return the status code “`404 Not Found`”.

[RSG-53] If a resource is deleted successfully, `DELETE` MUST return the status “`200 OK`” if the deleted resource is returned or “`204 No Content`” if it is not returned.

TRACE

45. The `TRACE` method does not carry API semantics and is used for testing and diagnostic information according to IETF RFC 2616, for example for testing a chain of proxies. `TRACE` allows the client to see what is being received at the other end of the request chain and uses that data. A `TRACE` request MUST NOT be idempotent according to the IETF RFC 2616

[RSG-54] The final recipient is either the origin server or the first proxy or gateway to receive a `Max-Forwards` value of zero in the request. A `TRACE` request MUST NOT include a body.

[RSG-55] A `TRACE` request MUST NOT be idempotent.

[RSG-56] The value of the `Via` HTTP header field MUST act to track the request chain.

[RSG-57] The `Max-Forwards` HTTP header field MUST be used to allow the client to limit the length of the request chain.

[RSG-58] If the request is valid, the response SHOULD contain the entire request message in the response body, with a `Content-Type` of "`message/http`".

[RSG-59] Responses to TRACE MUST NOT be cached.

[RSG-60] The status code "200 OK" SHOULD be returned to TRACE.

OPTIONS

46. When a client needs to learn information about a Web API, they can use OPTIONS. OPTIONS do not carry API semantics. An OPTIONS request MUST be idempotent according to the IETF RFC 2616, Custom HTTP Headers.

[RSG-61] An OPTIONS request MUST be idempotent.

47. It is a common practice for a Web API using custom HTTP headers to provide "X-" as a common prefix, which RFC 6648 deprecates and discourages to use.

[RSG-62] Custom HTTP headers starting with the "X-" prefix SHOULD NOT be used.

[RSG-63] Custom HTTP headers SHOULD NOT be used to change the behavior of HTTP Methods unless it is to resolve any existing technical limitations (for example, see [RSG-39]).

[RSG-64] The naming convention for custom HTTP headers is <organization>-<header name>, where <organization> and <header> SHOULD follow the kebab-case convention.

48. According to the service-oriented design principles, clients and services should evolve independently. Service versioning enables this. Common implementations of service versioning are: Header Versioning (by using a custom header), Query string versioning (for example, ?v=v1), Media type versioning (for example Accept: application/vnd.v1+json) and URI versioning (for example /api/v1/inventors).

[RSG-65] A Web API SHOULD support service versioning. URI versioning SHOULD be used for service versioning such as /v<version number> (for example /api/v1/inventors). Header Versioning, Query string versioning and Media type versioning SHOULD NOT be used.

49. According to the service-oriented design principles, service providers and consumers should also evolve independently. The service consumer should not be affected by minor (backward compatible) changes by the service provider. Therefore, service versioning should use only major versions. For internal non-published APIs (for example, for development and testing) minor versions may also be used such as Semantic Versioning.

[RSG-66] A versioning-numbering scheme SHOULD be followed considering only the major version number (for example /v1).

50. Service endpoint identifiers include information that can change over time. It may not be possible to replace all references to an out-of-date endpoint, which can lead to the service consumer being unable to further interact with the service endpoint. Therefore, the service provider may return a redirection response. The redirection may be temporary or permanent. The following HTTP status codes are available:

	Permanent	**Temporary**
Allows changing the request method from POST to GET	301	302
Doesn't allow changing the request method from POST to GET	308	307

Since 301 and 302 are more generic they are preferred to increase flexibility and overcome any unnecessary complexity.

[RSG-67] API service contracts MAY include endpoint redirection feature. When a service consumer attempts to invoke a service, a redirection response may be returned to tell the service consumer to resend the request to a new endpoint. Redirections MAY be temporary or permanent:

- Temporary redirect - using the HTTP response header Location and the HTTP status code "302 Found" according to IETF RFC 7231; or
- Permanent redirect - using the HTTP response header Location and the HTTP status code "301 Moved Permanently" according to IETF RFC 7238.

6.6. Data Query Patterns

Pagination Options

51. Pagination is a mechanism for a client to retrieve data in pages. Using pagination, we prevent overwhelming the service provider with resource demanding requests according to the design principles. The server should enforce a default page size in case the service consumer has not specified one. Paginated requests may not be idempotent, i.e., a paginated request does not create a snapshot of the data.

[RSG-68] A Web API SHOULD support pagination.

[RSG-69] Paginated requests MAY NOT be idempotent.

[RSG-70] A Web API MUST use query parameters to implement pagination.

[RSG-71] A Web API MUST NOT use HTTP headers to implement pagination.

[RSG-72] Query parameters `limit=<number of items to deliver>` and `offset=<number of items to skip>` SHOULD be used, where `limit` is the number of items to be returned (page size), and `skip` the number of items to be skipped (offset). If no page size limit is specified, a default SHOULD be defined - global or per collection; the default offset MUST be zero “0”. For example, the following is a valid URL:

```
https://wipo.int/api/v1/patents?limit=10&offset=20
```

[RSG-73] The `limit` and the `offset` parameter values SHOULD be included in the response.

Sorting

52. Retrieving data may require the data to be sorted by ascending or descending order. A multi-key sorting criterion may also be used. Sorting is determined through the use of the `sort` query string parameter. The value of this parameter is a comma-separated list of sort keys and sort directions that can optionally be appended to each sort key, separated by the colon ‘:’ character. The supported sort directions are either ‘`asc'` for ascending or ‘`desc'` for descending. The client may specify a sort direction for each key. If a sort direction is not specified for a key, then a default direction is set by the server.

For example:

(a) Only sort keys specified:

```
sort=key1,key2
```

‘key1’ is the first key and ‘key2’ is the second key and sort directions are defaulted by the server

(b) Some sort directions specified:

```
sort=key1:asc,key2
```

where `'key1'` is the first key (ascending order) and ‘`key2'` is the second key (direction defaulted by the server, i.e., any sort key without a corresponding direction is defaulted.).

(c) each keys with specified directions:

```
sort=key1:asc,key2:desc
```

where ‘`key1'` is the first key (ascending order) and ‘`key2'` is the second key (descending order).

53. In order to specify multi-attribute criteria sorting, the value of a query parameter may be a comma-separated list of sort keys and sort directions, with either ‘`asc`’ for ascending or ‘`desc`’ for descending which may be appended to each sort key, separated by the colon ‘:’ character.

[RSG-74] A Web API MUST support sorting.

[RSG-75] In order to specify a multi-attribute sorting criterion, a query parameter MUST be used. The value of this parameter is a comma-separated list of sort keys and sort directions either ‘`asc`’ for ascending or ‘`desc`’ for descending MAY be appended to each sort key, separated by the colon ‘:’ character. The default direction MUST be specified by the server in case that a sort direction is not specified for a key.

[RSG-76] A Web API SHOULD return the sorting criteria in the response.

Expand

54. A service consumer may control the amount of data it receives by expanding a single field into larger objects. Rather than simply asking for a linked entity ID to be included, a service caller can request the full representation of the entity be expanded within the results. Service calls may use expansions to get all the data they need in a single API request. For example:

```
GET https://wipo.int/api/v1/patents?id=id12345&expand=applicant
200 OK
{ "title": “Patent title”, “applicant”: {“name”:”applicant name”, …}, …}
In comparison to (if using hypermedia):
GET https://wipo.int/api/v1/patents?id=id12345
200 OK
{ "title": “Patent title”, “applicant”: {“href”:”
https://wipo.int/api/v1/link/to/applicants ”}, … }
```

55. A Web API may support expanding the body of returned content.

[RSG-77] A Web API MAY support expanding the body of returned content. The query parameter `expand=<comma-separated list of attributes names>` SHOULD be used.

Number of Items

56. In some use cases, the consumer of the API may be interested in the number of items in a collection. This is very common when combined with pagination in order to know the total number of items in the collection. For example,

```
GET https://wipo.int/api/v1/patents?count=true&limit=3&offset=4

200 OK

{"count": 100, ... }
```

57. As one alternative, a Web API may support returning the number of items in a collection inline, i.e., as the part of the response that contains the collection itself. Alternatively, it may form part of a metadata envelope, outside the main body of the response.

[RSG-78] A Web API MUST support returning the number of items in a collection.

[RSG-79] A query parameter MUST be used to support returning the number of items in a collection.

[RSG-80] The query parameter `count` SHOULD be used to return the number of items in a collection.

[RSG-81] A Web API MAY support returning the number of items in a collection inline, i.e., as the part of the response that contains the collection itself. A query parameter MUST be used.

[RSG-82] The query parameter `count=true` SHOULD be used. If not specified, `count` should be set by default to `false`.

[RSG-83] If a Web API supports pagination, it SHOULD support returning inline in the response the number of the collection (i.e., the total number of items of the collection).

Complex Search Expressions

58. For retrieving data with only a few search criteria, the query parameters are adequate. If there is a use case where we should search for data using complex search expressions (with multiple criteria, Boolean expressions and search operators) then the API has to be designed using a more complex query language. A query language has to be supported by a search grammar.

59. The Contextual Query Language (CQL) is a formal language for representing queries to information retrieval systems such as search engines, bibliographic catalogs and museum collection information. Based on the semantics of Z39.50[7], its design objective is that queries must be readable and writable and that the language is intuitive and maintains

Please refer the References chapter[7]

the expression of more complex query languages. This is just one option recommended for use, as it is used broadly by industry.

> [RSG-84] When a Web API supports complex search expressions then a query language SHOULD be specified, such as CQL.
>
> [RSG-85] A Service Contract MUST specify the grammar supported (such as fields, functions, keywords, and operators).
>
> [RSG-86] The query parameter “`q`” MUST be used.

6.7. Error Handling

60. Error responses should always use the appropriate HTTP status code selected from the standard list of HTTP status codes (RFC 7807), reproduced in Annex VII. When the requestor is expecting JSON, return error details in a common data structure. Unless the project requires otherwise, there is no need to define application-specific error codes. Stack trace and other debugging-related information should not be present in the error response body in production environments.

Error Payload

61. Error handling is carried out on two levels: on the protocol level (HTTP) and on the application level (payload returned). On the protocol level, a Web API returns an appropriate HTTP status code and on the application level, a Web API returns a payload reporting the error in adequate granularity (mandatory and optional attributes).

62. With regard to the mandatory and optional attributes for the application level error handling,

(a) the following `code` and `message` attributes are mandatory and while the `message` may change in the future, the `code` will not change; it is fixed and will always refer to this particular problem:

- `code` (integer) - Technical code of the error situation to be used for support purposes
- message (string) - User-facing (localizable) message describing the error request as requested by the HTTP header `Accept-Language`(see RS-112)

(b) The following attributes are conditionally mandatory:

- `details` - If error processing requires nesting of error responses, it must use the details field for this purpose. The details field must contain an array of JSON objects that shows code and message properties with the same semantics as described above.

(c) The following attributes are optional:

- `target` - The error structure may contain a target attribute that describes a data element (for example, a resource path).
- `status` - Duplicate of the HTTP status code to propagate it along the call chain or to write it in the support log without the need to explicitly add the HTTP status code every time.
- `moreInfo` - Array of links containing more information about the error situation, for example, giving hints to the end user.
- `internalMessage` - A technical message, for example, for logging purposes.

63. Error handling should follow HTTP standards (RFC 2616). A minimum error payload is recommended, for example for a JSON response:

```
404 Not Found
{
 "error": {
   "code": "03543762",
   "message": "Patent with ID 12345 not found",
   "target": "/api/v1/patents/12345",
   "details": [{
                 "code": "012312415",
                 "message": "Empty result set"
              }]
 }
}
```

[RSG-87] On the protocol level, a Web API MUST return an appropriate HTTP status code selected from the list of standard HTTP Status Codes.

[RSJ-88] On the application level, a Web API MUST return a payload reporting the error in adequate granularity. The `code` and `message` attributes are mandatory, the `details` attribute is conditionally mandatory and `target`, `status, moreInfo, and internalMessage` attributes are optional.

[RSG-89] Errors MUST NOT expose security-critical data or internal technical details, such as call stacks in the error messages.

[RSG-90] The HTTP `Header: Reason-Phrase` (described in RFC 2616) MUST NOT be used to carry error messages.

Correlation ID

64. Typically consuming a service cascades to triggering multiple other services. There should be a mechanism to correlate all the service activations in the same execution context. For example, including the correlation ID in the log messages, as this uniquely identifies the logged error.

[RSG-91] Every logged error SHOULD have a unique Correlation ID. A custom HTTP header SHOULD be used.

6.8. Service Contract

65. REST is not a protocol or an architecture, but an architectural style with architectural properties and architectural constraints. There are no official standards for REST API contracts. This Standard refers to API documentation as a REST Service Contract. The Service Contract is based on the following three fundamental elements:

(a) Resource identifier syntax – how can we express where the data is being transferred to or from?

(b) Methods – what are the protocol mechanisms used to transfer the data?

(c) Media types – what type of data is being transferred? Individual REST services use these elements in different combinations to expose their capabilities. Defining a master set of these elements for use by a collection (or inventory) of services makes this type of service contract "uniform".

[RSG-92] A Service Contract format MUST include the following:

- API version;
- Information about the semantics of API elements;
- Resources;
- Resource attributes;
- Query Parameters;
- Methods;
- Media types;
- Search grammar (if one is supported);
- HTTP Status Codes;
- HTTP Methods;
- Restrictions and distinctive features;
- Security (if any).

[RSG-93] A Service Contract format SHOULD include requests and responses in XML schema or JSON Schema and examples of the API usage in the supported formats, i.e., XML or JSON.

[RSG-94] A REST API MUST provide API documentation as a Service Contract.

[RSG-95] A Web API implementation deviating from this Standard MUST be explicitly documented in the Service Contract. If a deviating rule is not specified in the Service Contract, it MUST be assumed that this Standard is followed.

[RSG-96] A Service Contract MUST allow API client skeleton code generation.

[RSG-97] A Service Contract SHOULD allow server skeleton code generation.

66. Web API documentation can be written for example in RESTful API Modeling Language (RAML), Open API Specification (OAS) and WSDL. As only RAML fully supports both XML and JSON request/response validation (by using XSD schemas and JSON schemas), this Standard recommends RAML[8].

> [RSG-98] A Web API documentation SHOULD be written in RAML or OAS. Custom documentation formats SHOULD NOT be used.

6.9. Time-out

67. According to the service-oriented design principles, the server usage should be limited.

> [RSG-98] A Web API consumer SHOULD be able to specify a server timeout for each request; a custom HTTP header SHOULD be used. A maximum server timeout SHOULD be also used to protect server resources from over-use.

6.10. State Management

68. If development proceeds following the REST principles, state management must be dealt with on the client side, rather than on the server, since REST APIs are stateless. For example, if multiple servers implement a session, replication should be discouraged.

Response Versioning

69. Retrieving multiple times the same data set may result in bandwidth consumption if the data set has not been modified between the requests. Data should be conditionally be retrieved only if it has not been modified. This can be done with Content-based Resource Validation or Time-based Resource Validation. If using response versioning, a service consumer may implement optimistic locking.

> [RSG-99] A Web API SHOULD support conditionally retrieving data, to ensure only data which is modified will be retrieved. Content-based Resource Validation SHOULD be used because it is more accurate.
>
> [RSG-100] In order to implement Content-based Resource Validation the `ETag` HTTP header SHOULD be used in the response to encode the data state. Afterward, this value SHOULD be used in subsequent requests in the conditional HTTP headers (such as If-Match or If-None-Match). If the data has not been modified since the request returned the `ETag,` the server SHOULD return the status code “`304 Not Modified”` (if not modified). This mechanism is specified in IETF RFC 7231 and 7232.
>
> [RSG-101] In order to implement Time-based Resource Validation the `Last-Modified` HTTP header SHOULD be used. This mechanism is specified in IETF RFC 7231 and 7232.
>
> [RSG-102] Using response versioning, a service consumer MAY implement Optimistic Locking.

Caching

70. A Web API implementation should support cache handling in order to save bandwidth, in compliance with the IETF RFC 7234.

> [RSG-103] A Web API MUST support caching of `GET` results; a Web API MAY support caching of results from other HTTP Methods.
>
> [RSG-104] The HTTP response headers `Cache-Control` and `Expires` SHOULD be used. The latter MAY be used to support legacy clients.

Managed File Transfer

71. Transferring (i.e., downloading or uploading) large files has a high probability of causing a network interruption or some other transmission failure. It also consumes a large amount of memory for both the service provider and service consumer. Therefore, it is recommended to transfer large files in multiple chunks with multiple requests. This option also provides an indication of the total download or upload progress. The partial transfer of large files should resume support. The service provider should advertise if it supports the partial transfer of large files.[9]

72. There are two approaches for implementing this type of transfer: the first is to use a `Transfer-Encoding: chunked` header and the second using the `Content-Length` header. These headers should not be used together. `Content-Length` indicates the full size of the file transferred, and therefore the receiver will know the length of the body

[8] OAS is a specification. It also supports Markdown but RAML does not. On the other hand, although both OAS and RAML support JSON Schema validation for the requests and responses, OAS does not support XSDs. Therefore, in the future, when OAS is feature-complete it may be recommended.

[9] The service provider may return the location of the file and then the service consumer can call a directory service to download the file. At the end, a partial file download is required. This paragraph does not take into account non-REST protocols such as FTP or sFTP or rsync.

and will be able to estimate the download completion time. The `Transfer-Encoding: chunked` header is useful for streaming infinitely bounded data, such as audio or video, but not files. It is recommended to use the `Content-Length` header for downloading as the server utilization is low in comparison to `Transfer-Encoding: chunked.` For uploading, the `Transfer-Encoding: chunked` header is recommended.

A Web API should advertise if it supports partial file downloads by responding to `HEAD` requests and replying with the HTTP response headers: `Accept-Ranges` and `Content-Length`. The former should indicate the unit that can be used to define a range and should never be defined as' none'. The latter indicates the full size of the file to download.

[RSG-105] A Web API SHOULD advertise if it supports partial file downloads by responding to `HEAD` requests and replying with the HTTP response headers `Accept-Ranges` and `Content-Length`.

73. A Web API that supports downloading large files should support partial requests according to IETF RFC 7232, i.e.,:

- The service consumer asking for a range should use the HTTP header `Range`.
- The service provider response should contain the HTTP headers `Content-Range` and `Content-Length`.
- The service provider response should have the HTTP status `206 Partial Content` in case of a successful range request. In case of a range request that is out of bounds (range values overlap the extent of the resource), the server responds with a "`416 Requested Range Not Satisfiable`" status. In case range requested are not supported, the "`200 OK`" status is sent back from a server.

[RSG-106] A Web API SHOULD support partial file downloads. Multi-part ranges SHOULD be supported.

74. Multipart ranges may also be requested if the HTTP header `Content-Type: multipart/byteranges; boundary=XXXXX` is used. A range request may be conditional if it is combined with `ETag` or `If-Range` HTTP Headers.

75. There is not any IETF RFC for large files upload. Therefore, in this Standard we do not provide any implementation recommendation for large file uploads.

[RSG-107] A Web API SHOULD advertise if it supports partial file uploads.

[RSG-108] A Web API SHOULD support partial file uploaded. Multi-part ranges SHOULD be supported.

76. The IETF RFC 2616 does not impose any specific size limit for requests. The API Service Contract should specify the maximum limit for the requests. Moreover, on runtime the service provider should indicate to the service consumer if the allowed maximum limit has been exceeded.

[RSG-109] The service provider SHOULD return with HTTP response headers the HTTP header "`413 Request Entity Too Large`" in case the request has exceeded the maximum allowed limit. A custom HTTP header MAY be used to indicate the maximum size of the request.

6.11. Preference Handling

77. A service provider may allow a service consumer to configure values and influence how the former processes the requests of the latter. A standard means for implementing preference handling is outlined in IETF RFC 7240.

[RSG-110] If a Web API supports preference handling, it SHOULD be implemented according to IETF RFC 7240, i.e., the request HTTP header `Prefer` SHOULD be used and the response HTTP header `Preference-Applied` SHOULD be returned (echoing the original request).

[RSG-111] If a Web API supports preference handling, the nomenclature of preferences that MAY be set by using the `Prefer` header MUST be recorded in the Service Contract.

6.12. Translation

78. A service consumer may request responses in a specific language if the service provider supports it. A standard specification for handling of a set of natural languages is outlined in IETF TFC 7231.

[RSG-112] If a Web API supports localized data, the request HTTP header `Accept-Language` MUST be supported to indicate the set of natural languages that are preferred in the response as specified in IETF RFC 7231.

6.13. Long-Running Operations

79. There are cases, where a Web API may involve long running operations. For instance, the generation of a PDF by the service provider may take some minutes. This paragraph recommends a typical message exchange pattern to implement such cases, for example:

```
// (a)
```

```
GET https://wipo.int/api/v1/patents
Accept: application/pdf
…
// (b)
HTTP/1.1 202 Accepted
Location: https://wipo.int/api/v1/queues/12345
…
// (c1)
GET https://wipo.int/api/v1/queues/12345
…
HTTP/1.1 200 OK
…
// (c2)
GET https://wipo.int/api/v1/queues/12345
HTTP/1.1 303 See Other
Location: https://wipo.int/api/v1/path/to/pdf
…
// (c3)
GET https://wipo.int/api/v1/path/to/pdf
…
```

80. If an API supports long-running operations, then they should be performed asynchronously to ensure the user is not made to wait for a response. The rule below sets out a recommended approach for implementation.

[RSG-113] If the API supports long-running operations, they SHOULD be asynchronous. The following approach SHOULD be followed:

(a) The service consumer activates the service operation.

(b) The service operation returns the status code “`202 Accepted`” according to IETF RFC 7231 (section 6.3.3), i.e., the request has been accepted for processing but the processing has not been completed. The location of the queued task that was created is also returned with the HTTP header `Location`.

(c) The service consumer calls the returned `Location` to learn if the resource is available. If the resource is not available, the response SHOULD have the status code “`200 OK`”, contain the task status (for example pending) and MAY contain other information (for example, a link to cancel or delete the task using the DELETE HTTP method). If the resource is available, the response SHOULD have the status code “`303 See Other`” and the HTTP header `Location` SHOULD contain the URL to retrieve the task results.

6.14. Security Model

General Rules

81. Within the scope of this standard, API security is concerned with pivotal security attributes that will ensure that information accessible by an API and APIs themselves are secure throughout their lifecycle. These attributes are confidentiality, integrity, availability, trust, non-repudiation, compartmentalization, authentication, authorization and auditing.

[RSG-114] Confidentiality**:** APIs and API Information MUST be identified, classified, and protected against unauthorized access, disclosure and eavesdropping at all times. The least privilege, need to know and need to share[10] principles MUST be followed.

[RSG-115] Integrity-Assurance: APIs and API Information MUST be protected against unauthorized modification, duplication, corruption and destruction. Information MUST be modified through approved transactions and interfaces. Systems MUST be updated using approved configuration management, change management and patch management processes.

[RSG-116] Availability: APIs and API Information MUST be available to authorized users at the right time as defined in the Service Level Agreements (SLAs), access-control policies and defined business processes.

[RSG-117] Non-repudiation: Every transaction processed or action performed by APIs MUST enforce non-repudiation through the implementation of proper auditing, authorization, authentication, and the implementation of secure paths and non-repudiation services and mechanisms.

[RSG-118] Authentication, Authorization, Auditing: Users, systems, APIs or devices involved in critical transactions or actions MUST be authenticated, authorized using role-based or attribute based access-control services and maintain segregation of duty. In addition, all actions MUST be logged and the authentication’s strength must increase with the associated information risk.

“Security by Design Principles.” *OWASP*, https://www.owasp.org/index.php/Security_by_Design_Principles [10]

Guidelines for secure and threat-resistant API management

82. APIs should be designed, built, tested, and implemented with security requirements and risks in mind. The appropriate countermeasures and controls should be built directly into the design and not as an after-thought. It is recommended to use best practices and standards, such as OWASP.

[RSG-119] While developing APIs, threats, malicious use cases, secure coding techniques, transport layer security and security testing MUST be carefully considered, especially:

- `PUT`s and `POST`s – i.e.,: which change to internal data could potentially be used to attack or misinform.
- `DELETE`s – i.e.,: could be used to remove the contents of an internal resource repository
- Whitelist allowable methods- to ensure that allowable HTTP Methods are properly restricted while others would return a proper response code.
- Well known attacks should be considered during the threat-modeling phase of the design process to ensure that the threat risk does not increase. The threats and mitigation defined within OWASP Top Ten Cheat Sheet[11] MUST be taken into consideration.

[RSG-120] While developing APIs, the standards and best practices listed below SHOULD be followed:
Secure coding best practices: OWASP Secure Coding Principles

- Rest API security: REST Security Cheat Sheet
- Escape inputs and cross site scripting protection: OWASP XSS Cheat Sheet
- SQL Injection prevention: OWASP SQL Injection Cheat Sheet, OWASP Parameterization Cheat Sheet
- Transport layer security: OWASP Transport Layer Protection Cheat Sheet

[RSG-121] Security testing and vulnerability assessment MUST be carried out to ensure that APIs are secure and threat-resistant. This requirement MAY be achieved by leveraging Static and Dynamic Application Security Testing (SAST/DAST), automated vulnerability management tools and penetration testing.

Encryption, Integrity and non-repudiation

83. Protected services must be secured to protect authentication credentials in transit: for example passwords, API keys or JSON Web Tokens. Integrity of the transmitted data and non-repudiation of action taken should also be guaranteed. Secure cryptographic mechanisms can ensure confidentiality, encryption, integrity assurance and non-repudiation. Perfect forward secrecy is one means of ensuring that session keys cannot be compromised.

[RSG-122] Protected services MUST only provide HTTPS endpoints. TLS 1.2, or higher, with a cipher suite that includes ECDHE for key exchange.

[RSG-123] When considering authentication protocols, perfect forward secrecy SHOULD be used to provide transport security. The use of insecure cryptographic algorithms and backwards compatibility to SSL 3 and TLS 1.0/1.1 SHOULD NOT be allowed.

[RSG-124] For maximum security and trust, a site-to-site IPSEC VPN SHOULD be established to further protect the information transmitted over insecure networks.

[RSG-125] The consuming application SHOULD validate the TLS certificate chain when making requests to protected resources, including checking the certificate revocation list.

[RSG-126] Protected services SHOULD only use valid certificates issued by a trusted certificate authority (CA).

[RSG-127] Tokens SHOULD be signed using secure signing algorithms that are compliant with the digital signature standard (DSS) FIPS –186-4. The RSA digital signature algorithm or the ECDSA algorithm SHOULD be considered.

Authentication and Authorization

83. Authorization is the act of performing access control on a resource. Authorization does not just cover the enforcement of access controls, but also the definition of those controls. This includes the access rules and policies, which should define the required level of access agreeable to both provider and consuming application. The foundation of access control is a provider granting or denying a consuming application and/or consumer access to a resource to a certain level of granularity. Coarse-grained access should be considered at the API or the API gateway request point while fine-grained control should be considered at the backend service, if possible. Role Based Access Control (RBAC) or the Attribute Based Access Control (ABAC) model can be considered.

84. If a service is protected, then Open ID Connect should be favored over OAuth 2.0 because it fills many of the gaps of the latter and provides a standardized way to gain a resource owner's profile data, JSON Web Token (JWT) standardized

"Top 10-2017 Top 10." *OWASP*, http://www.owasp.org/index.php/Top_10-2017_Top_10 [11] token format and cryptography. Other security schemes should not be used such as HTTP Basic Authorization which requires that the client must keep a password somewhere in clear text to send along with each request. Also the verification of this password would be slower because it will have to access the credential store. OAuth 2.0 does not specify the security token. Therefore, the JWT token should be used in comparison for example to SAML 2.0, which is more verbose.

[RSG-128] Anonymous authentication MUST only be used when the customers and the application they are using accesses information or feature with a low sensitivity level which should not require authentication, such as, public information.

[RSG-129] Username and password or password hash authentication MUST NOT be allowed.

[RSG-130] If a service is protected, then Open ID Connect SHOULD be used.

[RSG-131] For use of JSON Web Tokens (JWT) consider the following:

- A JWT secret MUST possess high entropy to increase the work factor of a brute force attack.
- Token TTL and RTTL SHOULD be as short as possible.
- Sensitive information SHOULD not be stored in the JWT payload.

85. A common security design choice is to centralize user authentication. It should be stored in an Identity Provider (IdP) or locally at REST endpoints.

86. Services should be careful to prevent leaking of credentials. Passwords, security tokens, and API keys should not appear in the URL, as this can be captured in web server logs, which makes them intrinsically valuable. For example, the following is incorrect (API Key in URL): `https://wipo.int/api/patents?apiKey=a53f435643de32.`

[RSG-132] In `POST`/`PUT` requests, sensitive data SHOULD be transferred in the request body or by request headers.

[RSG-133] In `GET` requests, sensitive data SHOULD be transferred in an HTTP Header.

[RSG-134] In order to minimize latency and reduce coupling between protected services, the access control decision SHOULD be taken locally by REST endpoints.

87. API Keys Authentication: API keys should be used wherever system-to-system authentication is required API keys should be automatically and randomly generated. The inherent risk of this authentication mode is that anyone with a copy of the API key can use it as though they were the legitimate consuming application. Hence, all communications should be over TLS, to protect the key in transit. The onus is on the application developer to properly protect their copy of the API key. If the API key is embedded into the consuming application, it can be decompiled and extracted. If stored in plain text files, they can be stolen and re-used for malicious purposes. An API Key must therefore be protected by a credential store or a secret management mechanism. API Keys may be used to control services usage even for public services.

Certificate mutual authentication should be used when a Web API requires stronger authentication than offered by API keys and therefore overhead of public key cryptography and certificate are warranted. Secure and trusted certificates must be issued by a mutually trusted certificate authority (CA) through a trust establishment process or cross-certification.

[RSG-135] API Keys SHOULD be used for protected and public services to prevent overwhelming their service provider with multiple requests (denial-of-service attacks). For protected services API Keys MAY be used for monetization (purchased plans), usage policy enforcement (QoS) and monitoring.

[RSG-136] API Keys MAY be combined with the HTTP request header user-agent to discern between a human user and a software agent as specified in IETF RFC 7231.

[RSG-137] The service provider SHOULD return along with HTTP response headers the current usage status. The following response data MAY be returned:

- rate limit - rate limit (per minute) as set in the system;
- rate limit remaining - remaining amount of requests allowed during the current time slot (-1 indicates that the limit has been exceeded);
- rate limit reset - time (in seconds) remaining until the request counter will be reset.

[RSG-138] The service provider SHOULD return the status code “`429 Too Many Requests`” if requests are coming in too quickly.

[RSG-139] API Keys MUST be revoked if the client violates the usage agreement.

[RSG-140] API Keys SHOULD be transferred using custom HTTP headers. They SHOULD NOT be transferred using query parameters.

[RSG-141] API Keys SHOULD be randomly generated.

To mitigate identity security risks peculiar to sensitive systems and privileged actions, strong authentication can be leveraged. Certificates shared between the client and the server should be used, for example X.509.

> [RSG-142] For highly privileged services, two-way mutual authentication between the client and the server SHOULD use certificates to provide additional protection.
>
> [RSG-143] Multi-factor authentication SHOULD be implemented to mitigate identity risks for application with a high-risk profile, a system processing very sensitive information or a privileged action.

Availability and threat protection

88. Availability in this context covers threat protection to minimize API downtime, looking at how threats against exposed APIs can be mitigated using basic design principles. Availability also covers scaling to meet demand and ensuring the hosting environments are stable etc. These levels of availability are addressed across the hardware and software stacks that support the delivery of APIs. Availability is normally addressed under business continuity and disaster recovery standards that recommend a risk assessment approach to define the availability requirements.

Cross-domain Requests

89. Certain "cross-domain" requests, notably Ajax requests, are forbidden by default by the same-origin security policy. Under the same-origin policy, a web browser permits scripts contained in a first web page to access data in a second web page, only if both web pages have the same origin (i.e., combination of URI scheme, host name, and port number).

90. The Cross-Origin Resource Sharing (CORS) is a W3C standard to flexibly specify which Cross-Domain Requests are permitted. By delivering appropriate CORS HTTP headers, your REST API signals to the browser which domains or origins are allowed to make JavaScript calls to the REST service.

91. The JSON with padding (JSONP) is a method for sending JSON data without worrying about cross-domain request issues. It introduces callback functions for the loading of JSON data from different domains. The idea behind it is based on the fact that the HTML `<script>` tag is not affected by the same origin policy. Anything imported through this tag is executed immediately in the global context. Instead of passing in a JavaScript file, one can pass in a URL to a service that returns JavaScript code.

92. The following approaches are usually followed to bypass this restriction:

- JSONP is a workaround for cross-domain requests. It does not offer any error-detection mechanism, i.e., if there was an issue and the service failed or responded with an HTTP error, there is no way to determine what the issue was on the client side. The result will be that the AJAX application will just ‘hang’. Moreover, the site that uses JSONP will unconditionally trust the JSON provided from a different domain.
- Iframe is an alternative workaround for cross-domain requests. Using the JavaScript `window.postMessage (message, targetOrigin)` method on the iframe object, it is possible to pass a request a site of a different domain. Iframe approach has good compatibility even in old browsers. Moreover, it only supports GET. The source of the Iframes page should be always be checked due to security issues.
- CORS is a standardized approach to perform a call to an external domain. It can use `XMLHttpRequest` to send and receive data and has better error handling mechanism than JSONP. It supports many types of authorization in comparison to JSONP, which only supports cookies. It also supports HTTP Methods in comparison to JSONP, which only supports `GET`. On the other hand, it is not always possible to implement CORS because the browsers have to support it and because the API consumers have to be enlisted in the CORS whitelist.

> [RSG-144] If the REST API is public then the HTTP header `Access-Control-Allow-Origin` MUST be set to ‘*’.
>
> [RSG-145] If the REST API is protected then CORS SHOULD be used, if possible. Else, JSONP MAY be used as fallback but only for GET requests, for example, when the user is accessing using an old browser. Iframe SHOULD NOT be used.

6.15. API Maturity Model

93. It is common to classify a REST API using a maturity model. While various models are available, this Standard refers to the Richardson Maturity Model (RMM). RMM defines three levels and this Standard recommends Level 2 for REST API because Level 3 is complex to implement and requires significant conceptual and development-related investment from service providers and consumers. At the same time, it does not immediately benefit service consumers.

94. If a Web API implements Level 3 of RMM, a hypermedia format must be put in place. Hypertext Application Language (HAL)[12] is simple and is compatible with JSON and XML responses. However it is only a draft recommendation,

[12] “JSON Hypertext Application Language.” *IETF Tools*, https://tools.ietf.org/html/draft-kelly-json-hal-08

along with other hypermedia formats , such as JSON-LD[13]. JSON-Schema[14] should be used because as although there is currently no specification for Level 3 of RMM, this is considered the most mature. The following hypermedia formats should not be considered: IETF RFC 5988 and Collection+JSON.,

95. It is recommended that instances described by a schema provide a link to a downloadable JSON Schema using the link relation "describedby", as defined by Linked Data Protocol 1.0, section 8.1 [W3C.REC-ldp-20150226][15].
In HTTP, such links can be attached to any response using the `Link` header [**RFC8288**]. An example of such a header would be:

```
Link: <http://example.com/my-hyper-schema#>; rel="describedby"
```

[RSJ-146] If using instances described a schema, the `Link` header SHOULD be used to provide a link to a downloadable JSON schema ACCORDING TO RFC8288.

[RSJ-147] A Web API MUST implement at least Level 2 (Transport Native Properties) of RMM. Level 3 (Hypermedia) MAY be implemented to make the API completely discoverable.

96. A custom hypermedia format may be designed. In which case, a set of attributes is recommended. For example:

```
{
   "link": {
      "href": "/patents",
      "rel": "self"
   },
   ...
 }
```

[RSJ-148] For designing a custom hypermedia format the following set of attributes SHOULD be used enclosed into an attribute link:

– `href` – the target URI
– `rel` – the meaning of the target URI
– `self` – the URI references the resource itself
– `next` – the URI references the previous page (if used during pagination)
– `previous` – the URI references the next page (if used during pagination)
– arbitrary name v denotes the custom meaning of a relation.

7. SOAP WEB API

97. A SOAP Web API is a software application identified by URI, whose interfaces and binding are capable of being defined, described, and discovered by XML artifacts. It also supports direct interactions with other software applications using XML-based messages via internet protocols such as SOAP and HTTP.

98. A SOAP-based contract is described in a Web Service Definition Language (WSDL), a W3C standard document. Throughout this document “Web Service Contract WSDL document” will be referred as just “WSDL”.

99. When creating web services, there are two development styles: Contract Last and Contract First. When using a contract-last approach, you start with the code, and let the web service contract be generated from that. When using contract-first, you start with the WSDL contract, and use code to implement said contract.

7.1. General Rules

100. The Web Service Interoperability (WS-I) Profile is one of the most important standards in regards to SOAP-based APIs, and it provides a minimum foundation for writing Web Services that can work together. WS-I provides a guideline on how services are “exposed” to each other and how they transfer information (referred to as ‘messaging’). It is a profile for implementing specific versions of some of the most important Web Service standards such as WSDL, SOAP, XML, etc. Adhering to certain profiles implicitly indicates adhering to specific versions of these Web Services standards. WS-I Basic Profile v1.1 provides guidance for using XML 1.0, HTTP 1.1, UDDI, SOAP 1.1, WSDL 1.1, and UDDI 2.0. WS-I Basic Profile 2.0 provides guidance for using SOAP 1.2, WSDL 1.1, UDDI 2.0, WS-Addressing, and MTOM. SOAP 1.2 provides a clear

JSON-LD 1.0, https://www.w3.org/TR/json-ld/ [13]

[14] “Specification.” *JSON Schema*, https://json-schema.org/specification.html#specification-documents

“Abstract.” *JSON Schema: A Media Type for Describing JSON Documents*, https://json-schema.org/latest/json-schema-core.html#hypermedia [15] processing model and leads to better interoperability. WSDL 2.0 was designed to solve the interoperability issues found in WSDL 1.1 by using improved SOAP 1.2 bindings.

[WS-01] All WSDLs MUST conform to WS-I Basic Profile 2.0. WSDL 1.2 MAY be used.

101. A WSDL SOAP binding can be either a Remote Procedure Call (RPC) style binding or a document-style binding. A SOAP binding can also have an encoded use or a literal use. This gives you five style/use models: RPC/encoded, RPC/literal, document/encoded, document/literal, document/literal wrapped.

[WS-02] Services MUST follow document-style binding and literal use models (either document/literal or document/literal wrapped). When there are graphs, then the RPC/encoded style MUST be used.

[WS-03] When there are exceptional use cases, such as when there are overloaded operations in the WSDL, then all the other styles SHOULD be used.

102. The concrete WSDL should be separated from the abstract WSDL in order to provide a more modular and flexible interface. The abstract WSDL defines data types, messages, operation, and the port type. The concrete WSDL defines the binding, port and service.

[WS-04] The WSDL SHOULD be separated into an abstract and a concrete part.

[WS-05] All data types SHOULD be defined in an XSD file and imported in the abstract WSDL.

[WS-06] The concrete WSDL MUST define only one service with one port.

7.2. Schemas

103. Schemas used in the WSDL must be compliant with WIPO Standard ST.96 Standard. For re-use purposes and modularity, a schema must be a separate document that is either included or imported into the WSDL, instead of defining directly it in the WSDL. This will permit changes in XML structure without changing the WSDL.

[WS-07] The schema defined in the `wsdl:types` element MUST be imported from a self-standing schema file, to allow modularity and re-use.

[WS-08] Import of an external schema MUST be implemented using an `xsd:import` technique, not an `xsd:include`.

[WS-09] Element `xsd:any` MUST NOT be used to specify a root element in the message body.

[WS-10] The target namespace for the WSDL (attribute `targetNamespace` on `wsdl:definitions`) MUST be different from the target namespace of the schema (attribute `targetNamespace` on `xsd:schema`).

[WS-11] The requests and responses (naming convention, message format, data structure, and data dictionary) SHOULD follow WIPO Standard ST.96.

7.3. Naming and Versioning

104. Appropriate naming conventions should also be applied when naming Services and WSDL elements. Naming conventions should follow those implemented in WIPO Standard ST.96.

[WS-12] Services MUST be named in UpperCamelCase and have a 'Service' suffix, for example `https://wipo.int/PatentsService`.

[WS-13] WSDL elements message, part, portType, operation, input, output, and binding SHOULD be named in UpperCamelCase.

[WS-14] Request message names SHOULD have a ‘Request’ suffix.

[WS-15] Response message names SHOULD have a ‘Response` suffix.

[WS-16] Operation names SHOULD follow the format of `<Verb><Object>{<Qualifier>}`, where `<Verb>` indicates the operation (preferably Get, Create, Update, or Delete where applicable) on the `<Object>` of the operation, optionally finally followed by a `<Qualifier>` of the `<Object>`.

105. All operation names will have at least two parts. An optional third part may be included to further clarify and/or specify the business purpose of the operation. The three parts are: `<Verb> <Object> <Qualifier - Optional>`. Each part will be described in detail below.

Verb – Each operation name will start with a verb. The verb examples in common usage are described below:

Verb	Description	Example
`Get`	Get a single object	`GetBibData`
`Create`	Get a new object	`CreateBibData`
`Update`	Update an object	`UpdateBibData`
`Delete`	Delete an object	`DeleteCustomer`

Object – A noun following a verb will be a succinct and unambiguous description of the business function the operation is providing. The goal is to provide consumers with a better understanding of what the operation does with no ambiguity. Given that the definition of some entities are not common across the various cost centers, the object may be a composite field with the first node being the cost center and the second node the entity, for example, `PatentCustomer`.

Qualifier – The purpose of the object qualifier (optional) attribute is, to further clarify the business domain or subject area, for example, `GetCustomerList`. `Get` denotes the operation to be acted upon the Customer and `List` further describes the fact that the intention is to get a list of Customers not just one customer as in `GetCustomer`.

106. According to the service-oriented design principles, service providers and consumers should evolve independently. The service consumer should not be affected from minor (backward compatible) changes by the service provider. Therefore, service versioning should use only major version numbers. For internal APIs (for example, for development and testing) minor versions may also be used such as Semantic Versioning.

[WS-17] The name of the WSDL file SHOULD conform the following pattern: `<service name>_V<major version number>`

[WS-18] The namespace of the WSDL file SHOULD contain the service version; for example `https://wipo.int/PatentsService/V1”`

107. The description of service and its operations is provided as WSDL documentation.

[WS-19] Element `wsdl:documentation` SHOULD be used in WSDL with description of service (as the first child of `wsdl:definitions` in the WSDL) and its operations.

7.4. Web Service Contract Design

108. A Web Service Contract should include a technical interface comprised of a Web Service Definition Language (WSDL), XML Schema definitions, WS-Policy descriptions as well as a non-technical interface comprised of one or more service description documents.

109. The WSDL, part of the “Service Contract,” must be designed prior to any code development. No WSDL should ever be auto-generated from the code. The motto is “Contract First” and NOT “Code First”. All Web Service Contracts must conform to Web Service Interoperability Basic Profile (WS-I BP). Any project that auto-generates from code will be liable to amendments to ensure conformance to these standards.

7.5. Attaching Policies to WSDL Definitions

110. Web Service Contracts can be extended with security policies that express additional constraints, requirements, and qualities that typically relate to the behaviors of services. Security policies can be human-readable and become part of a supplemental service-level agreement, or can be machine-readable processed at runtime. Machine-readable policies are defined using the WS-Policy language and related WS-Policy specifications.

[WS-20] Policy expressions MUST be isolated into a separate WS-Policy definition document, which is then referenced within the WSDL document via the `wsp:PolicyReference` element.

[WS-21] Global or domain-specific policies SHOULD be isolated and applied to multiple services.

[WS-22] Policy attachment points SHOULD conform the WSDL 1.1 or later version, preferably version 2.0, attachment point elements and corresponding policy subjects (service, endpoint, operation, and message).

7.6. SOAP – Web Service Security

111. Web Services Security (WSS): SOAP Message Security is a set of enhancements to SOAP messaging that provides message integrity and confidentiality. WSS: SOAP Message Security is extensible, and can accommodate a variety of security models and encryption technologies. WSS: SOAP Message Security provides three main mechanisms that can be used independently or together:

- The ability to send security tokens as part of a message, and for associating the security tokens with message content
- The ability to protect the contents of a message from unauthorized and undetected modification (message integrity)
- The ability to protect the contents of a message from unauthorized disclosure (message confidentiality)

WSS: SOAP Message Security can be used in conjunction with other Web service extensions and application-specific protocols to satisfy a variety of security requirements.

[WS-23] Web Services using SOAP message SHOULD be protected accordance with WSS:SOAP Standard recommendations.

8. DATA TYPE FORMATS

112. This Standard recommends primitive data type formats such as time, date and language to be consistent with the recommendation of WIPO Standard ST.96 which are used both for XML and JSON requests and responses and for query parameters.

[CS-01] Time objects SHOULD be formatted as specified in IETF RFC 3339 (it is a profile of ISO 8601).

[CS-02] Time zone information SHOULD be used as specified in IETF RFC 3339. For example: `20:54:21+00:00`

[CS-03] Date objects SHOULD be formatted as specified in IETF RFC 3339 (it is a profile of ISO 8601). For example: `2018-10-19`

[CS-04] Datetime (i.e., timestamp) objects SHOULD be formatted as specified in IETF RFC 3339 (it is a profile of ISO 8601).

[CS-05] The relevant time zone SHOULD be used as specified in IETF RFC 3339. For example: `2017-02-14T20:54:21+00:00`

[CS-06] ISO 4217-Alpha (3-Letter Currency Codes) SHOULD be used for Currency Codes. The precision of the value (i.e., number of digits after the decimal point) MAY vary depending on the business requirements.

[CS-07] WIPO Standard ST.3 two-letter codes SHOULD be used for representing IPOs, states, other entities, organizations and for priority and designated countries/organizations.

[CS-08] ISO 3166-1-Alpha-2 Code Elements (2 letter country codes) SHOULD be used for the representation of the names of countries, dependencies, and other areas of particular geopolitical interest, on the basis of lists of country names obtained from the United Nations.

[CS-09] ISO 639-1 (2-Letter Language Codes) SHOULD be used for Language Codes.

[CS-10] Units of Measure SHOULD use the units of measure as described in The Unified Code for Units of Measure (based on ISO 80000 definitions). For example`, for weight measuring using kilograms (kg)`

[CSJ-11] Characters used in enumeration values MUST be restricted to the following set: {a-z, A-Z, 0-9, period (.), comma (,), spaces ( ), dash (-) and underscore (_).

[CSJ-12] The Representation Terms in Annex VIII SHOULD be used for atomic property names.

[CSJ-13] Acronyms and abbreviations appearing at the beginning of a property name SHOULD be in lower case. Otherwise all values of an enumeration, acronyms and abbreviation values MUST appear in upper case.

9. CONFORMANCE

113. This Standard is designed as a set of design rules and conventions that can be layered on top of existing or new Web Service APIs to provide common functionality. Not all services will support all of the conventions defined in the standard due to business (for example, QoS may not be required) or technical constraints (for example, OAuth 2.0 may already be used).

114. This standard defines two broad levels of conformance: A and AA Conformance Levels. Note that rules indicates as ‘MAY’ are not considered important when determining conformance with the standard.

115. The Web Service APIs are encouraged to support as much additional functionality beyond their level of conformance as is appropriate for their intended scenario.

116. Two broad conformance levels are defined:

- **Level A**: For Level A conformance, the API indicates that the required general design rules (RSG), which are identified as ‘MUST’ in this standard, are followed. In addition, the rules specific to the type of response returned must also be complied with, In other words, the following conformance sub-level are indicated:

    - Level AJ: returning a JSON response, must comply with all general level rules (RSG) identified as MUST as well as all JSON specific rules (RSJ) identified as MUST.

    - Level AX: returning an ST.96 XML instance, must comply with all general level rules (RSG) identified as MUST as well as all XML specific rules (RSX) identified as MUST.

- **Level AA:** For Level AA conformance, the API indicates that is Level A compliant and all the recommended design rules, which are identified as ‘SHOULD’ in this standard, are followed. As with Level A, there are sub-levels dependent upon the type of response:

    - Level AAJ: Level AJ compliance as well as the recommended SHOULD rules applicable to a JSON response.
    - Level AAX: Level AX compliance as well as the recommended SHOULD rules applicable to an XML response.

117. The traceability matric between the design rules and the conformance levels is listed in Annex I.

10. REFERENCES

WIPO Standards
WIPO ST.3 – “Two-letter codes for the representation of states, other entities and organizations”
WIPO ST.96 – “Processing of Industrial Property information using XML”

Standards and Conventions

- IEFT RFC 2119: Key words for use in RFCs to Indicate Requirement Levels – www.ietf.org/rfc/rfc2119.txt
- IEFT RFC 3339: Date and Time on the Internet: Timestamps – www.ietf.org/rfc/rfc3339.txt
- IEFT RFC 3986: Uniform Resource Identifier (URI): Generic Syntax – www.ietf.org/rfc/rfc3986.txt
- IEFT RFC 5789: PATCH Method for HTTP – https://tools.ietf.org/rfc/rfc5789.txt
- IEFT RFC 5988: Web Linking – https://tools.ietf.org/rfc/rfc5988.txt
- IEFT RFC 6648: Deprecating the "X-" Prefix and Similar Constructs in Application Protocols – https://tools.ietf.org/rfc/rfc6648.txt
- IEFT RFC 6750: The OAuth 2.0 Authorization Framework: Bearer Token Usage – https://tools.ietf.org/rfc/rfc6750.txt
- IEFT RFC 7231: Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content – www.ietf.org/rfc/rfc7231.txt
- IEFT RFC 7232: Hypertext Transfer Protocol (HTTP/1.1) – Conditional Requests www.ietf.org/rfc/rfc7232.txt
- IEFT RFC 7234: Hypertext Transfer Protocol (HTTP/1.1) – Caching www.ietf.org/rfc/rfc7234.txt
- IEFT RFC 7386: JSON Merge Patch – www.ietf.org/rfc/rfc7386.txt.
- IEFT RFC 7240: Prefer Header for HTTP – https://tools.ietf.org/rfc/rfc7240.txt
- IEFT RFC 7519: JSON Web Token – www.ietf.org/rfc/rfc7519.txt
- IEFT RFC 7540: Hypertext Transfer Protocol Version 2 (HTTP/2) – https://tools.ietf.org/html/rfc7540
- IEFT BCP-47: Tags for Identifying Languages – https://tools.ietf.org/rfc/bcp/bcp47.txt.
- ISO 639-1: Language codes – https://en.wikipedia.org/wiki/List_of_ISO_639-1_codes
- ISO 3166-1 alpha-2: Two-letter acronyms for country codes – https://en.wikipedia.org/wiki/ISO_3166-1_alpha-2
- ISO 3166-1 alpha-3: Three-letter acronyms for country codes – https://en.wikipedia.org/wiki/ISO_3166-1_alpha-3
- ISO 4217: Currency Codes – www.iso.org/iso/home/standards/currency_codes.htm
- ISO 8601: Date and Time Formats – https://en.wikipedia.org/wiki/ISO_8601
- OData - https://www.odata.org/
- OASIS OData Metadata Service Entity Model – http://docs.oasis-open.org/odata/odata/v4.0/os/models/MetadataService.edmx.
- OASIS OData JSON Format Version 4.0. Edited by Ralf Handl, Michael Pizzo, and Mark Biamonte. Latest version – http://docs.oasis-open.org/odata/odata-json-format/v4.0/odata-json-format-v4.0.html.
- OASIS OData Atom Format Version 4.0. Edited by Martin Zurmuehl, Michael Pizzo, and Ralf Handl. Latest version – http://docs.oasis-open.org/odata/odata-atom-format/v4.0/odata-atom-format-v4.0.html.
- OASIS OData "OData Version 4.0 Part 1: Protocol– http://docs.oasis-open.org/odata/odata/v4.0/os/part1-protocol/odata-v4.0-os-part1-protocol.html.

- OASIS OData Version 4.0 Part 2: URL Conventions – http://docs.oasis-open.org/odata/odata/v4.0/os/part2-url-conventions/odata-v4.0-os-part2-url-conventions.html.
- OASIS OData Version 4.0 Part 3: Common Schema Definition Language (CSDL) – http://docs.oasis-open.org/odata/odata/v4.0/os/part3-csdl/odata-v4.0-os-part3-csdl.html.
- OASIS ABNF components: OData ABNF Construction Rules Version 4.0 and OData ABNF Test Cases – http://docs.oasis-open.org/odata/odata/v4.0/os/abnf/
- OASIS Vocabulary components: OData Core Vocabulary, OData Measures Vocabulary and OData Capabilities Vocabulary – http://docs.oasis-open.org/odata/odata/v4.0/os/vocabularies/
- OASIS XML schemas: OData EDMX XML Schema and OData EDM XML Schema– http://docs.oasis-open.org/odata/odata/v4.0/os/schemas/
- OASIS SAML 2.0 – http://docs.oasis-open.org/security/saml/Post2.0/sstc-saml-tech-overview-2.0.html
- RAML (ReSTful API Modeling Language) – http://raml.org
- OpenAPI Initiative – www.openapis.org
- Richardson's REST API Maturity Model – https://martinfowler.com/articles/richardsonMaturityModel.html
- HAL – http://stateless.co/hal_specification.html
- JSON-LD – https://json-ld.org
- Collection+JSON - Document Format – http://amundsen.com/media-types/collection/format/
- BadgerFish – http://badgerfish.ning.com/
- Semantic Versioning – https://semver.org/
- REST – https://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm
- CQL – https://en.wikipedia.org/wiki/Contextual_Query_Language
- Z39.50 – https://www.loc.gov/z3950/agency/Z39-50-2003.pdf
- WS-I Basic Profile 2.0 – http://ws-i.org/profiles/basicprofile-2.0-2010-11-09.html
- W3C SOAP 1.2 Part 1: Messaging Framework – https://www.w3.org/TR/soap12-part1/
- W3C SOAP 1.2 Part 2: Adjuncts – https://www.w3.org/TR/soap12-part2/
- W3C WSDL Version 2.0 Part 1: Core Language – https://www.w3.org/TR/wsdl20/
- W3C CORS - https://www.w3.org/TR/cors/
- W3C Matric Parameters – https://www.w3.org/DesignIssues/MatrixURIs.html

IP Offices' REST APIs

- EPO – Open Patent Services OPS v 3.2 https://developers.epo.org
- USPTO – PatentsView http://www.patentsview.org/api/doc.html
- WIPO – ePCTv1.1 https://pct.wipo.int/
- EUIPO – TMview, Designview, TMclass http://www.tm-xml.org/TM-XML/TM-XML_xml/TM-XML_TM-Search.xml

Industry REST APIs and Design Guidelines

- Facebook – https://developers.facebook.com/docs/graph-api/reference
- GitHub – https://developer.github.com/v3
- Google APIs Design Guide – https://cloud.google.com/apis/design/
- Azure – https://docs.microsoft.com/en-us/rest/api/
- OpenAPI – https://swagger.io/docs/specification/about/
- OData – http://www.odata.org/documentation/
- JSON API – http://jsonapi.org/format/
- Microsoft API Design – https://docs.microsoft.com/en-us/azure/architecture/best-practices/api-design
- JIRA REST API – https://developer.atlassian.com/server/jira/platform/jira-rest-api-examples
- Confluece REST API – https://developer.atlassian.com/server/confluence/
- Ebay API – https://developer.ebay.com/api-docs/static/ebay-rest-landing.html
- Oracle REST Data Services – http://www.oracle.com/technetwork/developer-tools/rest-data-services/overview/index.html
- PayPal REST API – https://developer.paypal.com/docs/api/overview/
- Data on the Web Best Practices – https://www.w3.org/TR/dwbp/#intro
- SAP Guidelines for Future REST API Harmonization – https://d.dam.sap.com/m/xAUymP/54014_GB_54014_enUS.pdf
- GitHub API – https://developer.github.com/v3/
- Zalando – https://github.com/zalando/ReSTful-api-guidelines
- Dropbox – https://www.dropbox.com/developers
- Twitter – https://developer.twitter.com/en/docs

Others

- CQRS – https://martinfowler.com/bliki/CQRS.html
- ITU – https://www.itu.int/en/ITU-T/ipr/Pages/open.aspx
- OWASP Rest Security Cheat Sheet – https://www.owasp.org/index.php/REST_Security_Cheat_Sheet
- DDD – https://martinfowler.com/bliki/BoundedContext.html
- REST Principles – https://en.wikipedia.org/wiki/Representational_state_transfer
- Open/Closed Principle – https://en.wikipedia.org/wiki/Open/closed_principle
- Which style of WSDL should I use? – https://www.ibm.com/developerworks/library/ws-whichwsdl/
- https://www.ict.govt.nz/guidance-and-resources/standards-compliance/api-standard-and-guidelines/
- http://www.sabsa.org/node/69
- https://www.owasp.org/index.php/XSS_Prevention_Cheat_Sheet
- https://www.owasp.org/index.php/SQL_Injection_Prevention_Cheat_Sheet
- https://www.owasp.org/index.php/Security_by_Design_Principles
- https://www.owasp.org/index.php/OWASP_Top_Ten_Cheat_Sheet
- https://www.owasp.org/index.php/OWASP_API_Security_Project
- https://www.owasp.org/index.php/Input_Validation_Cheat_Sheet
- https://www.owasp.org/index.php/SQL_Injection_Prevention_Cheat_Sheet
- https://www.owasp.org/index.php/Query_Parameterization_Cheat_Sheet
- https://nvlpubs.nist.gov/nistpubs/fips/nist.fips.186-4.pdf
- http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soap-message-security-1.0.pdf
- SOA Principles of Service Design, Thomas Erl (2008)

[يلي ذلك المرفقات]

ANNEX I - LIST OF RESTFUL WEB SERVICE DESIGN RULES AND CONVENTIONS

The following tables summarize service design rules and conventions, and identifies basic conformance requirements in terms of which conformance level, Web Services API implementation support. In addition to the Rule ID and the Rule description, a cross reference is provided which indicates the other conformance levels that this rule is applicable to.

The following is a guide to the tables below:

- Table 1 provides a summary of rules that must be complied with in order to achieve a Level AJ compliance (for a JSON response);
- Table 2 provides a summary of design rules that must be complied with in order to achieve a Level AX compliance (for an XML response) ;
- Table 3 provides a summary of design rules that must be complied with in order to achieve a Level AAJ compliance (for a JSON response); and
- Table 4 provides s summary of design rules that must be complied with in order to achieve a Level AAX compliance (for an XML response).

[Note: Tables 1 to 4 remain incomplete until this new approach is approved by the CWS. An example is provided in each table.]

Table 1: Conformance Table JSON response (Level AJ)

Rule ID	Rule description	Cross reference
[RSG-01]	The forward slash character "/" MUST be used in the path of the URI to indicate a hierarchical relationship between resources but the path MUST NOT end with a forward slash as it does not provide any semantic value and may cause confusion.	AX, AAJ, AAX
[RSG-02]	Resources name MUST be consistent in their naming pattern.	
[RSG-04	Query parameters MUST be consistent in their naming pattern	
[RSG-06]	The URL pattern for a Web API MUST contain the word "api" in the URI.	
[RSG-07]	Matrix parameters MUST NOT be used.	
[RSG-08]	A Web API MUST consistently apply HTTP status codes as described in IETF RFCs	
[RSG-10]	If the API detects invalid input values, it MUST return the HTTP status code "400 Bad Request". The error payload MUST indicate the erroneous value.	
[RSG-12]	If the API detects valid values that require features to not be implemented, it MUST return the HTTP status code "501 Not Implemented". The error payload MUST indicate the unhandled value.	
[RSG-14]	If a resource can be stand-alone it MUST be a top-level resource, or otherwise a sub-resource.	
[RSG-15]	Query parameters MUST be used instead of URL paths to retrieve nested resources.	
[RSG-19]	Resource names, segment and query parameters MUST be composed of words in the English language, using the primary English spellings provided in the Oxford English Dictionary. Resource names that are localized due to business requirements MAY be in other languages.	

Rule ID	Rule description	Cross reference
[RSG-21]	A Web API MUST support content type negotiation following IETF RFC 7231.	
[RSG-22]	JSON format MUST be assumed when no specific content type is requested.	
[RSG-28]	A Web API MUST support at least XML or JSON.	
[RSG-29]	HTTP Methods MUST be restricted to the HTTP standard methods POST, GET, PUT, DELETE, OPTIONS, PATCH, TRACE and HEAD, as specified in IETF RFC 7231 and 5789.	
[RSG-34]	For an end point which fetches a single resource, if a resource is not found, the method GET MUST return the status code “404 Not Found”. Endpoints which return lists of resources will simply return an empty list.	
[RSG-35]	If a resource is retrieved successfully, the GET method MUST return 200 OK.	
[RSG-38]	A HEAD request MUST be idempotent.	
[RSG-40]	A POST request MUST NOT be idempotent according to the IETF RFC 2616.	
[RSG-44]	A PUT request MUST be idempotent.	
[RSG-45]	If a resource is not found, PUT MUST return the status code “404 Not Found”.	
[RSG-46]	If a resource is updated successfully, PUT MUST return the status code “200 OK” if the updated resource is returned or a “204 No Content” if it is not returned.	
[RSG-47]	A PATCH request MUST NOT be idempotent.	
[RSG-49]	If a resource is not found PATCH MUST return the status code “404 Not Found”.	
[RSJ-50]	If a Web API implements partial updates using PATCH, it MUST use the JSON Merge Patch format to describe the partial change set, as described in IETF RFC 7386 (by using the content type application/merge-patch+json.	
[RSG-51]	A DELETE request MUST NOT be idempotent.	
[RSG-52]	If a resource is not found, DELETE MUST return the status code “404 Not Found”.	
[RSG-53]	If a resource is deleted successfully, DELETE MUST return the status “200 OK” if the deleted resource is returned or “204 No Content” if it is not returned.	
[RSG-54]	The final recipient is either the origin server or the first proxy or gateway to receive a Max-Forwards value of zero in the request. A TRACE request MUST NOT include a body.	

Rule ID	Rule description	Cross reference
[RSG-55]	A TRACE request MUST NOT be idempotent.	
[RSG-56]	The value of the Via HTTP header field MUST act to track the request chain.	
[RSG-57]	The Max-Forwards HTTP header field MUST be used to allow the client to limit the length of the request chain.	
[RSG-59]	Responses to TRACE MUST NOT be cached.	
[RSG-61]	An OPTIONS request MUST be idempotent.	
[RSG-70]	A Web API MUST use query parameters to implement pagination.	
[RSG-71]	A Web API MUST NOT use HTTP headers to implement pagination.	
[RSG-74]	A Web API MUST support sorting.	
[RSG-75]	In order to specify a multi-attribute sorting criterion, a query parameter MUST be used. The value of this parameter is a comma-separated list of sort keys and sort directions either ‘asc’ for ascending or ‘desc’ for descending MAY be appended to each sort key, separated by the colon ‘:’ character. The default direction MUST be specified by the server in case that a sort direction is not specified for a key.	
[RSG-76]	A Web API SHOULD return the sorting criteria in the response.	
[RSG-78]	A Web API MUST support returning the number of items in a collection.	
[RSG-79]	A query parameter MUST be used to support returning the number of items in a collection.	
[RSG-81]	A Web API MAY support returning the number of items in a collection inline, i.e., as the part of the response that contains the collection itself. A query parameter MUST be used.	
[RSG-85]	A Service Contract MUST specify the grammar supported (such as fields, functions, keywords, and operators).	
[RSG-86]	The query parameter “q” MUST be used.	
[RSG-87]	On the protocol level, a Web API MUST return an appropriate HTTP status code selected from the list of standard HTTP Status Codes.	
[RSJ-88]	On the application level, a Web API MUST return a payload reporting the error in adequate granularity. The code and message attributes are mandatory, the details attribute is conditionally mandatory and target, status, moreInfo, and internalMessage attributes are optional.	
[RSG-89]	Errors MUST NOT expose security-critical data or internal technical details, such as call stacks in the error messages.	
[RSG-90]	The HTTP Header: Reason-Phrase (described in RFC 2616) MUST NOT be used to carry error messages.	

Rule ID	Rule description	Cross reference

[RSG-92]	A Service Contract format MUST include the following:
	– API version;
	– Information about the semantics of API elements;
	– Resources;
	– Resource attributes;
	– Query Parameters;
	– Methods;
	– Media types;
	– Search grammar (if one is supported);
	– HTTP Status Codes;
	– HTTP Methods;
	– Restrictions and distinctive features;
	– Security (if any).	

[RSG-94]	A REST API MUST provide API documentation as a Service Contract.	
[RSG-95]	A Web API implementation deviating from this Standard MUST be explicitly documented in the Service Contract. If a deviating rule is not specified in the Service Contract, it MUST be assumed that this Standard is followed.	
[RSG-96]	A Service Contract MUST allow API client skeleton code generation.	
[RSG-103]	A Web API MUST support caching of GET results; a Web API MAY support caching of results from other HTTP Methods.	
[RSG-111]	If a Web API supports preference handling, the nomenclature of preferences that MAY be set by using the Prefer header MUST be recorded in the Service Contract.	
[RSG-112]	If a Web API supports localized data, the request HTTP header Accept-Language MUST be supported to indicate the set of natural languages that are preferred in the response as specified in IETF RFC 7231.	
[RSG-114]	Confidentiality**:** APIs and API Information MUST be identified, classified, and protected against unauthorized access, disclosure and eavesdropping at all times. The least privilege, need to know and need to share[16] principles MUST be followed.	
[RSG-115]	Integrity-Assurance: APIs and API Information MUST be protected against unauthorized modification, duplication, corruption and destruction. Information MUST be modified through approved transactions and interfaces. Systems MUST be updated using approved configuration management, change management and patch management processes.	
[RSG-116]	Availability: APIs and API Information MUST be available to authorized users at the right time as defined in the Service Level Agreements (SLAs), access-control policies and defined business processes.	
[RSG-117]	Non-repudiation: Every transaction processed or action performed by APIs MUST enforce non-repudiation through the implementation of proper auditing, authorization, authentication, and the implementation of secure paths and non-repudiation services and mechanisms.	

Rule ID	Rule description	Cross reference
[RSG-118]	Authentication, Authorization, Auditing: Users, systems, APIs or devices involved in critical transactions or actions MUST be authenticated, authorized using role-based or attribute based access-control services and maintain segregation of duty. In addition, all actions MUST be logged and the authentication's strength must increase with the associated information risk.	
[RSG-119]	While developing APIs, threats, malicious use cases, secure coding techniques, transport layer security and security testing MUST be carefully considered, especially: – PUTs and POSTs – i.e.,: which change to internal data could potentially be used to attack or misinform. – DELETES – i.e.,: could be used to remove the contents of an internal resource repository – Whitelist allowable methods- to ensure that allowable HTTP Methods are properly restricted while others would return a proper response code. – Well known attacks should be considered during the threat-modeling phase of the design process to ensure that the threat risk does not increase. The threats and mitigation defined within OWASP Top Ten Cheat Sheet MUST be taken into consideration.	
[RSG-120]	Security testing and vulnerability assessment MUST be carried out to ensure that APIs are secure and threat-resistant. This requirement MAY be achieved by leveraging Static and Dynamic Application Security Testing (SAST/DAST), automated vulnerability management tools and penetration testing.	
[RSG-121]	Security testing and vulnerability assessment MUST be carried out to ensure that APIs are secure and threat-resistant. This requirement MAY be achieved by leveraging Static and Dynamic Application Security Testing (SAST/DAST), automated vulnerability management tools and penetration testing.	
[RSG-122]	Protected services MUST only provide HTTPS endpoints. TLS 1.2, or higher, with a cipher suite that includes ECDHE for key exchange.	
[RSG-128]	Anonymous authentication MUST only be used when the customers and the application they are using accesses information or feature with a low sensitivity level which should not require authentication, such as, public information.	
[RSG-129]	Username and password or password hash authentication MUST NOT be allowed.	
[RSG-139]	API Keys MUST be revoked if the client violates the usage agreement.	
[RSG-144]	If the REST API is public then the HTTP header Access-Control-Allow-Origin MUST be set to '*'.	
[RSJ-147]	A Web API MUST implement at least Level 2 (Transport Native Properties) of RMM. Level 3 (Hypermedia) MAY be implemented to make the API completely discoverable.	

Table 2: Conformance Table XML response (Level AX)

Rule ID	Rule description	Cross reference
[RSG-01]	The forward slash character "/" MUST be used in the path of the URI to indicate a hierarchical relationship between resources but the path	

Rule ID	Rule description	Cross reference
	MUST NOT end with a forward slash as it does not provide any semantic value and may cause confusion.	AJ, AAJ, AAX
[RSG-02]	Resources name MUST be consistent in their naming pattern.	
[RSG-04	Query parameters MUST be consistent in their naming pattern	
[RSG-06]	The URL pattern for a Web API MUST contain the word “api” in the URI.	
[RSG-07]	Matrix parameters MUST NOT be used.	
[RSG-08]	A Web API MUST consistently apply HTTP status codes as described in IETF RFCs	
[RSG-10]	If the API detects invalid input values, it MUST return the HTTP status code “400 Bad Request”. The error payload MUST indicate the erroneous value.	
[RSG-12]	If the API detects valid values that require features to not be implemented, it MUST return the HTTP status code “501 Not Implemented”. The error payload MUST indicate the unhandled value.	
[RSG-14]	If a resource can be stand-alone it MUST be a top-level resource, or otherwise a sub-resource.	
[RSG-15]	Query parameters MUST be used instead of URL paths to retrieve nested resources.	
[RSG-19]	Resource names, segment and query parameters MUST be composed of words in the English language, using the primary English spellings provided in the Oxford English Dictionary. Resource names that are localized due to business requirements MAY be in other languages.	
[RSG-21]	A Web API MUST support content type negotiation following IETF RFC 7231.	
[RSG-22]	JSON format MUST be assumed when no specific content type is requested.	
[RSG-28]	A Web API MUST support at least XML or JSON.	
[RSG-29]	HTTP Methods MUST be restricted to the HTTP standard methods POST, GET, PUT, DELETE, OPTIONS, PATCH, TRACE and HEAD, as specified in IETF RFC 7231 and 5789.	
[RSG-34]	For an end point which fetches a single resource, if a resource is not found, the method GET MUST return the status code “404 Not Found”. Endpoints which return lists of resources will simply return an empty list.	
[RSG-35]	If a resource is retrieved successfully, the GET method MUST return 200 OK.	
[RSG-38]	A HEAD request MUST be idempotent.	

Rule ID	Rule description	Cross reference
[RSG-40]	A POST request MUST NOT be idempotent according to the IETF RFC 2616.	
[RSG-44]	A PUT request MUST be idempotent.	
[RSG-45]	If a resource is not found, PUT MUST return the status code “404 Not Found”.	
[RSG-46]	If a resource is updated successfully, PUT MUST return the status code “200 OK” if the updated resource is returned or a “204 No Content” if it is not returned.	
[RSG-47]	A PATCH request MUST NOT be idempotent.	
[RSG-49]	If a resource is not found PATCH MUST return the status code “404 Not Found”.	
[RSG-51]	A DELETE request MUST NOT be idempotent.	
[RSG-52]	If a resource is not found, DELETE MUST return the status code “404 Not Found”.	
[RSG-53]	If a resource is deleted successfully, DELETE MUST return the status “200 OK” if the deleted resource is returned or “204 No Content” if it is not returned.	
[RSG-54]	The final recipient is either the origin server or the first proxy or gateway to receive a Max-Forwards value of zero in the request. A TRACE request MUST NOT include a body.	
[RSG-55]	A TRACE request MUST NOT be idempotent.	
[RSG-56]	The value of the Via HTTP header field MUST act to track the request chain.	
[RSG-57]	The Max-Forwards HTTP header field MUST be used to allow the client to limit the length of the request chain.	
[RSG-59]	Responses to TRACE MUST NOT be cached.	
[RSG-61]	An OPTIONS request MUST be idempotent.	
[RSG-70]	A Web API MUST use query parameters to implement pagination.	
[RSG-71]	A Web API MUST NOT use HTTP headers to implement pagination.	
[RSG-74]	A Web API MUST support sorting.	
[RSG-75]	In order to specify a multi-attribute sorting criterion, a query parameter MUST be used. The value of this parameter is a comma-separated list of sort keys and sort directions either ‘asc’ for ascending or ‘desc’ for descending MAY be appended to each sort key, separated by the colon ‘:’ character. The default direction MUST be specified by the server in case that a sort direction is not specified for a key.	

Rule ID	Rule description	Cross reference
[RSG-76]	A Web API SHOULD return the sorting criteria in the response.	
[RSG-78]	A Web API MUST support returning the number of items in a collection.	
[RSG-79]	A query parameter MUST be used to support returning the number of items in a collection.	
[RSG-81]	A Web API MAY support returning the number of items in a collection inline, i.e., as the part of the response that contains the collection itself. A query parameter MUST be used.	
[RSG-85]	A Service Contract MUST specify the grammar supported (such as fields, functions, keywords, and operators).	
[RSG-86]	The query parameter "q" MUST be used.	
[RSG-87]	On the protocol level, a Web API MUST return an appropriate HTTP status code selected from the list of standard HTTP Status Codes.	
[RSJ-88]	On the application level, a Web API MUST return a payload reporting the error in adequate granularity. The code and message attributes are mandatory, the details attribute is conditionally mandatory and target, status, moreInfo, and internalMessage attributes are optional.	
[RSG-89]	Errors MUST NOT expose security-critical data or internal technical details, such as call stacks in the error messages.	
[RSG-90]	The HTTP Header: Reason-Phrase (described in RFC 2616) MUST NOT be used to carry error messages.	

[RSG-92]	A Service Contract format MUST include the following:
	– API version;
	– Information about the semantics of API elements;
	– Resources;
	– Resource attributes;
	– Query Parameters;
	– Methods;
	– Media types;
	– Search grammar (if one is supported);
	– HTTP Status Codes;
	– HTTP Methods;
	– Restrictions and distinctive features;
	– Security (if any).	

[RSG-94]	A REST API MUST provide API documentation as a Service Contract.	
[RSG-95]	A Web API implementation deviating from this Standard MUST be explicitly documented in the Service Contract. If a deviating rule is not specified in the Service Contract, it MUST be assumed that this Standard is followed.	
[RSG-96]	A Service Contract MUST allow API client skeleton code generation.	
[RSG-103]	A Web API MUST support caching of GET results; a Web API MAY support caching of results from other HTTP Methods.	

Rule ID	Rule description	Cross reference
[RSG-111]	If a Web API supports preference handling, the nomenclature of preferences that MAY be set by using the Prefer header MUST be recorded in the Service Contract.	
[RSG-112]	If a Web API supports localized data, the request HTTP header Accept-Language MUST be supported to indicate the set of natural languages that are preferred in the response as specified in IETF RFC 7231.	
[RSG-114]	Confidentiality**:** APIs and API Information MUST be identified, classified, and protected against unauthorized access, disclosure and eavesdropping at all times. The least privilege, need to know and need to share principles MUST be followed.	
[RSG-115]	Integrity-Assurance: APIs and API Information MUST be protected against unauthorized modification, duplication, corruption and destruction. Information MUST be modified through approved transactions and interfaces. Systems MUST be updated using approved configuration management, change management and patch management processes.	
[RSG-116]	Availability: APIs and API Information MUST be available to authorized users at the right time as defined in the Service Level Agreements (SLAs), access-control policies and defined business processes.	
[RSG-117]	Non-repudiation: Every transaction processed or action performed by APIs MUST enforce non-repudiation through the implementation of proper auditing, authorization, authentication, and the implementation of secure paths and non-repudiation services and mechanisms.	
[RSG-118]	Authentication, Authorization, Auditing: Users, systems, APIs or devices involved in critical transactions or actions MUST be authenticated, authorized using role-based or attribute based access-control services and maintain segregation of duty. In addition, all actions MUST be logged and the authentication's strength must increase with the associated information risk.	
[RSG-119]	While developing APIs, threats, malicious use cases, secure coding techniques, transport layer security and security testing MUST be carefully considered, especially: – PUTs and POSTs – i.e.,: which change to internal data could potentially be used to attack or misinform. – DELETES – i.e.,: could be used to remove the contents of an internal resource repository – Whitelist allowable methods- to ensure that allowable HTTP Methods are properly restricted while others would return a proper response code. – Well known attacks should be considered during the threat-modeling phase of the design process to ensure that the threat risk does not increase. The threats and mitigation defined within OWASP Top Ten Cheat Sheet MUST be taken into consideration.	
[RSG-120]	Security testing and vulnerability assessment MUST be carried out to ensure that APIs are secure and threat-resistant. This requirement MAY be achieved by leveraging Static and Dynamic Application Security Testing (SAST/DAST), automated vulnerability management tools and penetration testing.	
[RSG-121]	Security testing and vulnerability assessment MUST be carried out to ensure that APIs are secure and threat-resistant. This requirement MAY be achieved by leveraging Static and Dynamic Application Security	

Rule ID	Rule description	Cross reference
	Testing (SAST/DAST), automated vulnerability management tools and penetration testing.	
[RSG-122]	Protected services MUST only provide HTTPS endpoints. TLS 1.2, or higher, with a cipher suite that includes ECDHE for key exchange.	
[RSG-128]	Anonymous authentication MUST only be used when the customers and the application they are using accesses information or feature with a low sensitivity level which should not require authentication, such as, public information.	
[RSG-129]	Username and password or password hash authentication MUST NOT be allowed.	
[RSG-139]	API Keys MUST be revoked if the client violates the usage agreement.	
[RSG-144]	If the REST API is public then the HTTP header Access-Control-Allow-Origin MUST be set to ‘*’.	
[RSJ-147]	A Web API MUST implement at least Level 2 (Transport Native Properties) of RMM. Level 3 (Hypermedia) MAY be implemented to make the API completely discoverable.	

Table 3: Conformance Table Level AAJ (JSON Response)

Rule ID	Rule description	Cross reference
[RSG-01]	The forward slash character “/” MUST be used in the path of the URI to indicate a hierarchical relationship between resources but the path MUST NOT end with a forward slash as it does not provide any semantic value and may cause confusion.	AJ, AX, AAX
[RSG-02]	Resources name MUST be consistent in their naming pattern.	
[RSG-03]	Resource names SHOULD use lowercase or kebab-case naming conventions. Resources name MAY be abbreviated.	
[RSG-05]	Query parameters SHOULD use the lowerCamelCase convention. Query parameter MAY be abbreviated.	
[RSG-06]	The URL pattern for a Web API MUST contain the word “api” in the URI.	
[RSG-07]	Matrix parameters MUST NOT be used.	
[RSG-08]	A Web API MUST consistently apply HTTP status codes as described in IETF RFCs	
[RSG-09]	The recommended codes in Annex VI SHOULD be used by a Web API to classify the error.	
[RSG-10]	If the API detects invalid input values, it MUST return the HTTP status code “400 Bad Request”. The error payload MUST indicate the erroneous value.	
[RSG-11]	If the API detects syntactically correct argument names (in the request or query parameters) that are not expected, it SHOULD ignore them.	
[RSG-12]	If the API detects valid values that require features to not be implemented, it MUST return the HTTP status code “501 Not Implemented”. The error payload MUST indicate the unhandled value.	

Rule ID	Rule description	Cross reference
[RSG-13]	A Web API SHOULD only use top-level resources. If there are sub-resources, they should be collections and imply an association. An entity should be accessible as either top-level resource or sub-resource but not using both ways.	
[RSG-14]	If a resource can be stand-alone it MUST be a top-level resource, or otherwise a sub-resource.	
[RSG-15]	Query parameters MUST be used instead of URL paths to retrieve nested resources.	
[RSG-16]	A query parameter SHOULD be used instead of URL paths in case that a Web API supports projection following the format: “fields=”<comma-separated list of attribute names>.	
[RSG-17]	Resource names SHOULD be nouns for CRUD Web APIs and verbs for Intent Web APIs.	
[RSG-18]	If resource name is a noun it SHOULD always use the plural form. Irregular noun forms SHOULD NOT be used. For example, /persons should be used instead of /people.	
[RSG-19]	Resource names, segment and query parameters MUST be composed of words in the English language, using the primary English spellings provided in the Oxford English Dictionary. Resource names that are localized due to business requirements MAY be in other languages.	
[RSG-20]	A Web API SHOULD use for content type negotiation the request HTTP header Accept and the response HTTP header Content-Type.	
[RSG-21]	A Web API MUST support content type negotiation following IETF RFC 7231.	
[RSG-22]	JSON format MUST be assumed when no specific content type is requested.	
[RSG-23]	A Web API SHOULD return the status code “406 Not Acceptable” if a requested format is not supported.	
[RSG-24]	A Web API SHOULD reject requests containing unexpected or missing content type headers with the HTTP status code “406 Not Acceptable” or “415 Unsupported Media Type”.	
[RSJ-26]	JSON object property names SHOULD be provided in lowerCamelCase, e.g., applicantName.	
[RSG-28]	A Web API MUST support at least XML or JSON.	
[RSG-29]	HTTP Methods MUST be restricted to the HTTP standard methods POST, GET, PUT, DELETE, OPTIONS, PATCH, TRACE and HEAD, as specified in IETF RFC 7231 and 5789.	
[RSG-31]	Some proxies support only POST and GET methods. To overcome these limitations, a Web API MAY use a POST method with a custom HTTP header “tunneling” the real HTTP method. The custom HTTP header X-HTTP-Method SHOULD be used.	
[RSG-32]	If a HTTP Method is not supported, the HTTP status code “405 Method Not Allowed” SHOULD be returned.	
[RSG-33]	A Web API SHOULD support batching operations (aka bulk operations) in place of multiple individual requests to achieve latency reduction. The	

Rule ID	Rule description	Cross reference
	same semantics should be used for HTTP Methods and HTTP status codes. The response payload SHOULD contain information about all batching operations. If multiple errors occur, the error payload SHOULD contain information about all the occurrences (in the details attribute). All bulk operations SHOULD be executed in an atomic operation.	
[RSG-34]	For an end point which fetches a single resource, if a resource is not found, the method GET MUST return the status code “404 Not Found”. Endpoints which return lists of resources will simply return an empty list.	
[RSG-35]	If a resource is retrieved successfully, the GET method MUST return 200 OK.	
[RSG-37]	When the URI length exceeds the 255 bytes, then the POST method SHOULD be used instead of GET due to GET limitations, or else create named queries if possible.	
[RSG-38]	A HEAD request MUST be idempotent.	
[RSG-39]	Some proxies support only POST and GET methods. A Web API SHOULD support a custom HTTP request header to override the HTTP Method in order to overcome these limitations.	
[RSG-40]	A POST request MUST NOT be idempotent according to the IETF RFC 2616.	
[RSG-41]	If the resource creation was successful, the HTTP header Location SHOULD contain a URI (absolute or relative) pointing to a created resource.	
[RSG-42]	If the resource creation was successful, the response SHOULD contain the status code “201 Created”.	
[RSG-43]	If the resource creation was successful, the response payload SHOULD by default contain the body of the created resource, to allow the client to use it without making an additional HTTP call.	
[RSG-44]	A PUT request MUST be idempotent.	
[RSG-45]	If a resource is not found, PUT MUST return the status code “404 Not Found”.	
[RSG-46]	If a resource is updated successfully, PUT MUST return the status code “200 OK” if the updated resource is returned or a “204 No Content” if it is not returned.	
[RSG-47]	A PATCH request MUST NOT be idempotent.	
[RSG-48]	If a Web API implements partial updates, idempotent characteristics of PATCH SHOULD be taken into account. In order to make it idempotent the API MAY follow the IETF RFC 5789 suggestion of using optimistic locking.	
[RSG-49]	If a resource is not found PATCH MUST return the status code “404 Not Found”.	
[RSJ-50]	If a Web API implements partial updates using PATCH, it MUST use the JSON Merge Patch format to describe the partial change set, as described in IETF RFC 7386 (by using the content type application/merge-patch+json.	

Rule ID	Rule description	Cross reference
[RSG-51]	A DELETE request MUST NOT be idempotent.	
[RSG-52]	If a resource is not found, DELETE MUST return the status code “404 Not Found”.	
[RSG-53]	If a resource is deleted successfully, DELETE MUST return the status “200 OK” if the deleted resource is returned or “204 No Content” if it is not returned.	
[RSG-54]	The final recipient is either the origin server or the first proxy or gateway to receive a Max-Forwards value of zero in the request. A TRACE request MUST NOT include a body.	
[RSG-55]	A TRACE request MUST NOT be idempotent.	
[RSG-56]	The value of the Via HTTP header field MUST act to track the request chain.	
[RSG-57]	The Max-Forwards HTTP header field MUST be used to allow the client to limit the length of the request chain.	
[RSG-58]	If the request is valid, the response SHOULD contain the entire request message in the response body, with a Content-Type of "message/http".	
[RSG-59]	Responses to TRACE MUST NOT be cached.	
[RSG-60]	The status code “200 OK” SHOULD be returned to TRACE.	
[RSG-61]	An OPTIONS request MUST be idempotent.	
[RSG-62]	Custom HTTP headers starting with the “X-” prefix SHOULD NOT be used.	
[RSG-63]	Custom HTTP headers SHOULD NOT be used to change the behavior of HTTP Methods unless it is to resolve any existing technical limitations (for example, see [RSG-39]).	
[RSG-64]	The naming convention for custom HTTP headers is <organization>-<header name>, where <organization> and <header> SHOULD follow the kebab-case convention.	
[RSG-65]	A Web API SHOULD support service versioning. URI versioning SHOULD be used for service versioning such as /v<version number> (for example /api/v1/inventors). Header Versioning, Query string versioning and Media type versioning SHOULD NOT be used.	
[RSG-66]	A versioning-numbering scheme SHOULD be followed considering only the major version number (for example /v1).	
[RSG-68]	A Web API SHOULD support pagination.	
[RSG-70]	A Web API MUST use query parameters to implement pagination.	
[RSG-71]	A Web API MUST NOT use HTTP headers to implement pagination.	
[RSG-72]	Query parameters limit=<number of items to deliver> and offset=<number of items to skip> SHOULD be used, where limit is the number of items to be returned (page size), and skip the number of items to be skipped (offset). If no page size limit is specified, a default SHOULD be defined - global or per collection; the default offset MUST be zero “0”. For example, the following is a valid URL:	

Rule ID	Rule description	Cross reference
	https://wipo.int/api/v1/patents?limit=10&offset=20	
[RSG-73]	The limit and the offset parameter values SHOULD be included in the response.	
[RSG-74]	A Web API MUST support sorting.	
[RSG-75]	In order to specify a multi-attribute sorting criterion, a query parameter MUST be used. The value of this parameter is a comma-separated list of sort keys and sort directions either ‘asc’ for ascending or ‘desc’ for descending MAY be appended to each sort key, separated by the colon ‘:’ character. The default direction MUST be specified by the server in case that a sort direction is not specified for a key.	
[RSG-76]	A Web API SHOULD return the sorting criteria in the response.	
[RSG-77]	A Web API MAY support expanding the body of returned content. The query parameter expand=<comma-separated list of attributes names> SHOULD be used.	
[RSG-78]	A Web API MUST support returning the number of items in a collection.	
[RSG-79]	A query parameter MUST be used to support returning the number of items in a collection.	
[RSG-80]	The query parameter count SHOULD be used to return the number of items in a collection.	
[RSG-81]	A Web API MAY support returning the number of items in a collection inline, i.e., as the part of the response that contains the collection itself. A query parameter MUST be used.	
[RSG-82]	The query parameter count=true SHOULD be used. If not specified, count should be set by default to false.	
[RSG-83]	If a Web API supports pagination, it SHOULD support returning inline in the response the number of the collection (i.e., the total number of items of the collection).	
[RSG-84]	When a Web API supports complex search expressions then a query language SHOULD be specified, such as CQL.	
[RSG-85]	A Service Contract MUST specify the grammar supported (such as fields, functions, keywords, and operators).	
[RSG-86]	The query parameter “q” MUST be used.	
[RSG-87]	On the protocol level, a Web API MUST return an appropriate HTTP status code selected from the list of standard HTTP Status Codes.	
[RSJ-88]	On the application level, a Web API MUST return a payload reporting the error in adequate granularity. The code and message attributes are mandatory, the details attribute is conditionally mandatory and target, status, moreInfo, and internalMessage attributes are optional.	
[RSG-89]	Errors MUST NOT expose security-critical data or internal technical details, such as call stacks in the error messages.	
[RSG-90]	The HTTP Header: Reason-Phrase (described in RFC 2616) MUST NOT be used to carry error messages.	

Rule ID	Rule description	Cross reference
[RSG-91]	Every logged error SHOULD have a unique Correlation ID. A custom HTTP header SHOULD be used.	

[RSG-92]	A Service Contract format MUST include the following:
	– API version;
	– Information about the semantics of API elements;
	– Resources;
	– Resource attributes;
	– Query Parameters;
	– Methods;
	– Media types;
	– Search grammar (if one is supported);
	– HTTP Status Codes;
	– HTTP Methods;
	– Restrictions and distinctive features;
	– Security (if any).	

[RSG-93]	A Service Contract format SHOULD include the following: – Schemas validating the requests and responses (for example, XSD and JSON Schema); – Examples of the API usage should be provided in all the supported formats (for example, XML and JSON).	
[RSG-94]	A REST API MUST provide API documentation as a Service Contract.	
[RSG-95]	A Web API implementation deviating from this Standard MUST be explicitly documented in the Service Contract. If a deviating rule is not specified in the Service Contract, it MUST be assumed that this Standard is followed.	
[RSG-96]	A Service Contract MUST allow API client skeleton code generation.	
[RSG-97]	A Service Contract SHOULD allow server skeleton code generation.	
[RSG-98]	A Web API documentation SHOULD be written in RAML or OAS. Custom documentation formats SHOULD NOT be used.	
[RSG-99]	A Web API SHOULD support conditionally retrieving data, to ensure only data which is modified will be retrieved. Content-based Resource Validation SHOULD be used because it is more accurate.	
[RSG-100]	In order to implement Content-based Resource Validation the ETag HTTP header SHOULD be used in the response to encode the data state. Afterward, this value SHOULD be used in subsequent requests in the conditional HTTP headers (such as If-Match or If-None-Match). If the data has not been modified since the request returned the ETag, the server SHOULD return the status code “304 Not Modified” (if not modified). This mechanism is specified in IETF RFC 7231 and 7232.	
[RSG-101]	In order to implement Time-based Resource Validation the Last-Modified HTTP header SHOULD be used. This mechanism is specified in IETF RFC 7231 and 7232.	

Rule ID	Rule description	Cross reference
[RSG-103]	A Web API MUST support caching of GET results; a Web API MAY support caching of results from other HTTP Methods.	
[RSG-104]	The HTTP response headers Cache-Control and Expires SHOULD be used. The latter MAY be used to support legacy clients.	
[RSG-105]	A Web API SHOULD advertise if it supports partial file downloads by responding to HEAD requests and replying with the HTTP response headers Accept-Ranges and Content-Length.	
[RSG-106]	A Web API SHOULD support partial file downloads. Multi-part ranges SHOULD be supported.	
[RSG-107]	A Web API SHOULD advertise if it supports partial file uploads.	
[RSG-108]	A Web API SHOULD support partial file uploaded. Multi-part ranges SHOULD be supported.	
[RSG-109]	The service provider SHOULD return with HTTP response headers the HTTP header “413 Request Entity Too Large” in case the request has exceeded the maximum allowed limit. A custom HTTP header MAY be used to indicate the maximum size of the request.	
[RSG-110]	If a Web API supports preference handling, it SHOULD be implemented according to IETF RFC 7240, i.e., the request HTTP header Prefer SHOULD be used and the response HTTP header Preference-Applied SHOULD be returned (echoing the original request).	
[RSG-111]	If a Web API supports preference handling, the nomenclature of preferences that MAY be set by using the Prefer header MUST be recorded in the Service Contract.	
[RSG-112]	If a Web API supports localized data, the request HTTP header Accept-Language MUST be supported to indicate the set of natural languages that are preferred in the response as specified in IETF RFC 7231.	
[RSG-113]	If the API supports long-running operations, they SHOULD be asynchronous. The following approach SHOULD be followed: The service consumer activates the service operation. The service operation returns the status code “202 Accepted” according to IETF RFC 7231 (section 6.3.3), i.e., the request has been accepted for processing but the processing has not been completed. The location of the queued task that was created is also returned with the HTTP header Location. The service consumer calls the returned Location to learn if the resource is available. If the resource is not available, the response SHOULD have the status code “200 OK”, contain the task status (for example pending) and MAY contain other information (for example, a link to cancel or delete the task using the DELETE HTTP method). If the resource is available, the response SHOULD have the status code “303 See Other” and the HTTP header Location SHOULD contain the URL to retrieve the task results.	
[RSG-114]	Confidentiality**:** APIs and API Information MUST be identified, classified, and protected against unauthorized access, disclosure and eavesdropping at all times. The least privilege, need to know and need to share principles MUST be followed.	
[RSG-115]	Integrity-Assurance: APIs and API Information MUST be protected against unauthorized modification, duplication, corruption and	

Rule ID	Rule description	Cross reference
	destruction. Information MUST be modified through approved transactions and interfaces. Systems MUST be updated using approved configuration management, change management and patch management processes.	
[RSG-116]	Availability: APIs and API Information MUST be available to authorized users at the right time as defined in the Service Level Agreements (SLAs), access-control policies and defined business processes.	
[RSG-117]	Non-repudiation: Every transaction processed or action performed by APIs MUST enforce non-repudiation through the implementation of proper auditing, authorization, authentication, and the implementation of secure paths and non-repudiation services and mechanisms.	
[RSG-118]	Authentication, Authorization, Auditing: Users, systems, APIs or devices involved in critical transactions or actions MUST be authenticated, authorized using role-based or attribute based access-control services and maintain segregation of duty. In addition, all actions MUST be logged and the authentication's strength must increase with the associated information risk.	
[RSG-119]	While developing APIs, threats, malicious use cases, secure coding techniques, transport layer security and security testing MUST be carefully considered, especially: – PUTs and POSTs – i.e.,: which change to internal data could potentially be used to attack or misinform. – DELETES – i.e.,: could be used to remove the contents of an internal resource repository – Whitelist allowable methods- to ensure that allowable HTTP Methods are properly restricted while others would return a proper response code. – Well known attacks should be considered during the threat-modeling phase of the design process to ensure that the threat risk does not increase. The threats and mitigation defined within OWASP Top Ten Cheat Sheet MUST be taken into consideration.	
[RSG-120]	While developing APIs, the standards and best practices listed below SHOULD be followed: – Secure coding best practices: OWASP Secure Coding Principles – Rest API security: REST Security Cheat Sheet – Escape inputs and cross site scripting protection: OWASP XSS Cheat Sheet – SQL Injection prevention: OWASP SQL Injection Cheat Sheet, OWASP Parameterization Cheat Sheet Transport layer security: OWASP Transport Layer Protection Cheat Sheet	
[RSG-121]	Security testing and vulnerability assessment MUST be carried out to ensure that APIs are secure and threat-resistant. This requirement MAY be achieved by leveraging Static and Dynamic Application Security Testing (SAST/DAST), automated vulnerability management tools and penetration testing.	
[RSG-122]	Protected services MUST only provide HTTPS endpoints. TLS 1.2, or higher, with a cipher suite that includes ECDHE for key exchange.	
[RSG-123]	When considering authentication protocols, perfect forward secrecy SHOULD be used to provide transport security. The use of insecure	

Rule ID	Rule description	Cross reference
	cryptographic algorithms and backwards compatibility to SSL 3 and TLS 1.0/1.1 SHOULD NOT be allowed.	
[RSG-124]	For maximum security and trust, a site-to-site IPSEC VPN SHOULD be established to further protect the information transmitted over insecure networks.	
[RSG-125]	The consuming application SHOULD validate the TLS certificate chain when making requests to protected resources, including checking the certificate revocation list.	
[RSG-126]	Protected services SHOULD only use valid certificates issued by a trusted certificate authority (CA).	
[RSG-127]	Tokens SHOULD be signed using secure signing algorithms that are compliant with the digital signature standard (DSS) FIPS –186-4. The RSA digital signature algorithm or the ECDSA algorithm SHOULD be considered.	
[RSG-128]	Anonymous authentication MUST only be used when the customers and the application they are using accesses information or feature with a low sensitivity level which should not require authentication, such as, public information.	
[RSG-129]	Username and password or password hash authentication MUST NOT be allowed.	
[RSG-130]	If a service is protected, then Open ID Connect SHOULD be used.	
[RSG-131]	For use of JSON Web Tokens (JWT) consider the following: – A JWT secret MUST possess high entropy to increase the work factor of a brute force attack. – Token TTL and RTTL SHOULD be as short as possible. – Sensitive information SHOULD not be stored in the JWT payload. – [RSG-130] In POST/PUT requests, sensitive data SHOULD be transferred in the request body or by request headers. – [RSG-131] In GET requests, sensitive data SHOULD be transferred in an HTTP Header. – [RSG-132] In order to minimize latency and reduce coupling between protected services, the access control decision SHOULD be taken locally by REST endpoints.	
[RSG-132]	In POST/PUT requests, sensitive data SHOULD be transferred in the request body or by request headers.	
[RSG-133]	In GET requests, sensitive data SHOULD be transferred in an HTTP Header.	
[RSG-134]	In order to minimize latency and reduce coupling between protected services, the access control decision SHOULD be taken locally by REST endpoints.	
[RSG-135]	API Keys SHOULD be used for protected and public services to prevent overwhelming their service provider with multiple requests (denial-of-service attacks). For protected services API Keys MAY be used for monetization (purchased plans), usage policy enforcement (QoS) and monitoring.	

Rule ID	Rule description	Cross reference
[RSG-137]	The service provider SHOULD return along with HTTP response headers the current usage status. The following response data MAY be returned: – rate limit - rate limit (per minute) as set in the system; – rate limit remaining - remaining amount of requests allowed during the current time slot (-1 indicates that the limit has been exceeded); – rate limit reset - time (in seconds) remaining until the request counter will be reset.	
[RSG-138]	The service provider SHOULD return the status code “429 Too Many Requests” if requests are coming in too quickly.	
[RSG-139]	API Keys MUST be revoked if the client violates the usage agreement.	
[RSG-140]	API Keys SHOULD be transferred using custom HTTP headers. They SHOULD NOT be transferred using query parameters.	
[RSG-141]	API Keys SHOULD be randomly generated.	
[RSG-142]	For highly privileged services, two-way mutual authentication between the client and the server SHOULD use certificates to provide additional protection.	
[RSG-143]	Multi-factor authentication SHOULD be implemented to mitigate identity risks for application with a high-risk profile, a system processing very sensitive information or a privileged action.	
[RSG-144]	If the REST API is public then the HTTP header Access-Control-Allow-Origin MUST be set to ‘*’.	
[RSG-145]	If the REST API is protected then CORS SHOULD be used, if possible. Else, JSONP MAY be used as fallback but only for GET requests, for example, when the user is accessing using an old browser. Iframe SHOULD NOT be used.	
[RSJ-146]	If using instances described a schema, the Link header SHOULD be used to provide a link to a downloadable JSON schema ACCORDING TO RFC8288.	
[RSJ-147]	A Web API MUST implement at least Level 2 (Transport Native Properties) of RMM. Level 3 (Hypermedia) MAY be implemented to make the API completely discoverable.	
[RSJ-148]	For designing a custom hypermedia format the following set of attributes SHOULD be used enclosed into an attribute link: – `href` – the target URI – `rel` – the meaning of the target URI – `self` – the URI references the resource itself – `next` – the URI references the previous page (if used during pagination) – `previous` – the URI references the next page (if used during pagination) – arbitrary name v denotes the custom meaning of a relation.	

Table 4: Conformance Level AAX

Rule ID	Rule description	Cross reference
[RSG-01]	The forward slash character "/" MUST be used in the path of the URI to indicate a hierarchical relationship between resources but the path MUST NOT end with a forward slash as it does not provide any semantic value and may cause confusion.	AJ, AX, AAJ
[RSG-02]	Resources name MUST be consistent in their naming pattern.	
[RSG-03]	Resource names SHOULD use lowercase or kebab-case naming conventions. Resources name MAY be abbreviated.	
[RSG-05]	Query parameters SHOULD use the lowerCamelCase convention. Query parameter MAY be abbreviated.	
[RSG-06]	The URL pattern for a Web API MUST contain the word "api" in the URI.	
[RSG-07]	Matrix parameters MUST NOT be used.	
[RSG-08]	A Web API MUST consistently apply HTTP status codes as described in IETF RFCs	
[RSG-09]	The recommended codes in Annex VI SHOULD be used by a Web API to classify the error.	
[RSG-10]	If the API detects invalid input values, it MUST return the HTTP status code "400 Bad Request". The error payload MUST indicate the erroneous value.	
[RSG-11]	If the API detects syntactically correct argument names (in the request or query parameters) that are not expected, it SHOULD ignore them.	
[RSG-12]	If the API detects valid values that require features to not be implemented, it MUST return the HTTP status code "501 Not Implemented". The error payload MUST indicate the unhandled value.	
[RSG-13]	A Web API SHOULD only use top-level resources. If there are sub-resources, they should be collections and imply an association. An entity should be accessible as either top-level resource or sub-resource but not using both ways.	
[RSG-14]	If a resource can be stand-alone it MUST be a top-level resource, or otherwise a sub-resource.	
[RSG-15]	Query parameters MUST be used instead of URL paths to retrieve nested resources.	
[RSG-16]	A query parameter SHOULD be used instead of URL paths in case that a Web API supports projection following the format: "fields="<comma-separated list of attribute names>.	
[RSG-17]	Resource names SHOULD be nouns for CRUD Web APIs and verbs for Intent Web APIs.	
[RSG-18]	If resource name is a noun it SHOULD always use the plural form. Irregular noun forms SHOULD NOT be used. For example, /persons should be used instead of /people.	
[RSG-19]	Resource names, segment and query parameters MUST be composed of words in the English language, using the primary English spellings provided in the Oxford English Dictionary. Resource names that are localized due to business requirements MAY be in other languages.	
[RSG-20]	A Web API SHOULD use for content type negotiation the request HTTP header Accept and the response HTTP header Content-Type.	

Rule ID	Rule description	Cross reference
[RSG-21]	A Web API MUST support content type negotiation following IETF RFC 7231.	
[RSG-22]	JSON format MUST be assumed when no specific content type is requested.	
[RSG-23]	A Web API SHOULD return the status code “406 Not Acceptable” if a requested format is not supported.	
[RSG-24]	A Web API SHOULD reject requests containing unexpected or missing content type headers with the HTTP status code “406 Not Acceptable” or “415 Unsupported Media Type”.	
[RSX-25]	The requests and responses (naming convention, message format, data structure, and data dictionary) SHOULD refer to WIPO Standard ST.96.	
[RSX-27]	XML components SHOULD be provided in UpperCamelCase in line with WIPO Standard ST.96.	
[RSG-28]	A Web API MUST support at least XML or JSON.	
[RSG-29]	HTTP Methods MUST be restricted to the HTTP standard methods POST, GET, PUT, DELETE, OPTIONS, PATCH, TRACE and HEAD, as specified in IETF RFC 7231 and 5789.	
[RSG-31]	Some proxies support only POST and GET methods. To overcome these limitations, a Web API MAY use a POST method with a custom HTTP header “tunneling” the real HTTP method. The custom HTTP header X-HTTP-Method SHOULD be used.	
[RSG-32]	If a HTTP Method is not supported, the HTTP status code “405 Method Not Allowed” SHOULD be returned.	
[RSG-33]	A Web API SHOULD support batching operations (aka bulk operations) in place of multiple individual requests to achieve latency reduction. The same semantics should be used for HTTP Methods and HTTP status codes. The response payload SHOULD contain information about all batching operations. If multiple errors occur, the error payload SHOULD contain information about all the occurrences (in the details attribute). All bulk operations SHOULD be executed in an atomic operation.	
[RSG-34]	For an end point which fetches a single resource, if a resource is not found, the method GET MUST return the status code “404 Not Found”. Endpoints which return lists of resources will simply return an empty list.	
[RSG-35]	If a resource is retrieved successfully, the GET method MUST return 200 OK.	
[RSG-37]	When the URI length exceeds the 255 bytes, then the POST method SHOULD be used instead of GET due to GET limitations, or else create named queries if possible.	
[RSG-38]	A HEAD request MUST be idempotent.	
[RSG-39]	Some proxies support only POST and GET methods. A Web API SHOULD support a custom HTTP request header to override the HTTP Method in order to overcome these limitations.	
[RSG-40]	A POST request MUST NOT be idempotent according to the IETF RFC 2616.	
[RSG-41]	If the resource creation was successful, the HTTP header Location SHOULD contain a URI (absolute or relative) pointing to a created resource.	
[RSG-42]	If the resource creation was successful, the response SHOULD contain the status code “201 Created”.	

Rule ID	Rule description	Cross reference
[RSG-43]	If the resource creation was successful, the response payload SHOULD by default contain the body of the created resource, to allow the client to use it without making an additional HTTP call.	
[RSG-44]	A PUT request MUST be idempotent.	
[RSG-45]	If a resource is not found, PUT MUST return the status code “404 Not Found”.	
[RSG-46]	If a resource is updated successfully, PUT MUST return the status code “200 OK” if the updated resource is returned or a “204 No Content” if it is not returned.	
[RSG-47]	A PATCH request MUST NOT be idempotent.	
[RSG-48]	If a Web API implements partial updates, idempotent characteristics of PATCH SHOULD be taken into account. In order to make it idempotent the API MAY follow the IETF RFC 5789 suggestion of using optimistic locking.	
[RSG-49]	If a resource is not found PATCH MUST return the status code “404 Not Found”.	
[RSG-51]	A DELETE request MUST NOT be idempotent.	
[RSG-52]	If a resource is not found, DELETE MUST return the status code “404 Not Found”.	
[RSG-53]	If a resource is deleted successfully, DELETE MUST return the status “200 OK” if the deleted resource is returned or “204 No Content” if it is not returned.	
[RSG-54]	The final recipient is either the origin server or the first proxy or gateway to receive a Max-Forwards value of zero in the request. A TRACE request MUST NOT include a body.	
[RSG-55]	A TRACE request MUST NOT be idempotent.	
[RSG-56]	The value of the Via HTTP header field MUST act to track the request chain.	
[RSG-57]	The Max-Forwards HTTP header field MUST be used to allow the client to limit the length of the request chain.	
[RSG-58]	If the request is valid, the response SHOULD contain the entire request message in the response body, with a Content-Type of "message/http".	
[RSG-59]	Responses to TRACE MUST NOT be cached.	
[RSG-60]	The status code “200 OK” SHOULD be returned to TRACE.	
[RSG-61]	An OPTIONS request MUST be idempotent.	
[RSG-62]	Custom HTTP headers starting with the “X-” prefix SHOULD NOT be used.	
[RSG-63]	Custom HTTP headers SHOULD NOT be used to change the behavior of HTTP Methods unless it is to resolve any existing technical limitations (for example, see [RSG-39]).	
[RSG-64]	The naming convention for custom HTTP headers is <organization>-<header name>, where <organization> and <header> SHOULD follow the kebab-case convention.	
[RSG-65]	A Web API SHOULD support service versioning. URI versioning SHOULD be used for service versioning such as /v<version number> (for example /api/v1/inventors).	

Rule ID	Rule description	Cross reference
	Header Versioning, Query string versioning and Media type versioning SHOULD NOT be used.	
[RSG-66]	A versioning-numbering scheme SHOULD be followed considering only the major version number (for example /v1).	
[RSG-68]	A Web API SHOULD support pagination.	
[RSG-70]	A Web API MUST use query parameters to implement pagination.	
[RSG-71]	A Web API MUST NOT use HTTP headers to implement pagination.	
[RSG-72]	Query parameters limit=<number of items to deliver> and offset=<number of items to skip> SHOULD be used, where limit is the number of items to be returned (page size), and skip the number of items to be skipped (offset). If no page size limit is specified, a default SHOULD be defined - global or per collection; the default offset MUST be zero “0”. For example, the following is a valid URL: https://wipo.int/api/v1/patents?limit=10&offset=20	
[RSG-73]	The limit and the offset parameter values SHOULD be included in the response.	
[RSG-74]	A Web API MUST support sorting.	
[RSG-75]	In order to specify a multi-attribute sorting criterion, a query parameter MUST be used. The value of this parameter is a comma-separated list of sort keys and sort directions either ‘asc’ for ascending or ‘desc’ for descending MAY be appended to each sort key, separated by the colon ‘:’ character. The default direction MUST be specified by the server in case that a sort direction is not specified for a key.	
[RSG-76]	A Web API SHOULD return the sorting criteria in the response.	
[RSG-77]	A Web API MAY support expanding the body of returned content. The query parameter expand=<comma-separated list of attributes names> SHOULD be used.	
[RSG-78]	A Web API MUST support returning the number of items in a collection.	
[RSG-79]	A query parameter MUST be used to support returning the number of items in a collection.	
[RSG-80]	The query parameter count SHOULD be used to return the number of items in a collection.	
[RSG-81]	A Web API MAY support returning the number of items in a collection inline, i.e., as the part of the response that contains the collection itself. A query parameter MUST be used.	
[RSG-82]	The query parameter count=true SHOULD be used. If not specified, count should be set by default to false.	
[RSG-83]	If a Web API supports pagination, it SHOULD support returning inline in the response the number of the collection (i.e., the total number of items of the collection).	
[RSG-84]	When a Web API supports complex search expressions then a query language SHOULD be specified, such as CQL.	
[RSG-85]	A Service Contract MUST specify the grammar supported (such as fields, functions, keywords, and operators).	
[RSG-86]	The query parameter “q” MUST be used.	

Rule ID	Rule description	Cross reference
[RSG-87]	On the protocol level, a Web API MUST return an appropriate HTTP status code selected from the list of standard HTTP Status Codes.	
[RSJ-88]	On the application level, a Web API MUST return a payload reporting the error in adequate granularity. The code and message attributes are mandatory, the details attribute is conditionally mandatory and target, status, moreInfo, and internalMessage attributes are optional.	
[RSG-89]	Errors MUST NOT expose security-critical data or internal technical details, such as call stacks in the error messages.	
[RSG-90]	The HTTP Header: Reason-Phrase (described in RFC 2616) MUST NOT be used to carry error messages.	

[RSG-92]	A Service Contract format MUST include the following:
	– API version;
	– Information about the semantics of API elements;
	– Resources;
	– Resource attributes;
	– Query Parameters;
	– Methods;
	– Media types;
	– Search grammar (if one is supported);
	– HTTP Status Codes;
	– HTTP Methods;
	– Restrictions and distinctive features;
	– Security (if any).	

[RSG-93]	A Service Contract format SHOULD include the following: – Schemas validating the requests and responses (for example, XSD and JSON Schema); – Examples of the API usage should be provided in all the supported formats (for example, XML and JSON).	
[RSG-94]	A REST API MUST provide API documentation as a Service Contract.	
[RSG-95]	A Web API implementation deviating from this Standard MUST be explicitly documented in the Service Contract. If a deviating rule is not specified in the Service Contract, it MUST be assumed that this Standard is followed.	
[RSG-96]	A Service Contract MUST allow API client skeleton code generation.	
[RSG-97]	A Service Contract SHOULD allow server skeleton code generation.	
[RSG-98]	A Web API documentation SHOULD be written in RAML or OAS. Custom documentation formats SHOULD NOT be used.	
[RSG-99]	A Web API SHOULD support conditionally retrieving data, to ensure only data which is modified will be retrieved. Content-based Resource Validation SHOULD be used because it is more accurate.	
[RSG-100]	In order to implement Content-based Resource Validation the ETag HTTP header SHOULD be used in the response to encode the data state. Afterward, this value SHOULD be used in subsequent requests in the conditional HTTP headers (such as If-Match or If-None-Match). If the data has not been modified since the request	

Rule ID	Rule description	Cross reference
	returned the ETag, the server SHOULD return the status code “304 Not Modified” (if not modified). This mechanism is specified in IETF RFC 7231 and 7232.	
[RSG-101]	In order to implement Time-based Resource Validation the Last-Modified HTTP header SHOULD be used. This mechanism is specified in IETF RFC 7231 and 7232.	
[RSG-104]	The HTTP response headers Cache-Control and Expires SHOULD be used. The latter MAY be used to support legacy clients.	
[RSG-105]	A Web API SHOULD advertise if it supports partial file downloads by responding to HEAD requests and replying with the HTTP response headers Accept-Ranges and Content-Length.	
[RSG-106]	A Web API SHOULD support partial file downloads. Multi-part ranges SHOULD be supported.	
[RSG-107]	A Web API SHOULD advertise if it supports partial file uploads.	
[RSG-108]	A Web API SHOULD support partial file uploaded. Multi-part ranges SHOULD be supported.	
[RSG-109]	The service provider SHOULD return with HTTP response headers the HTTP header “413 Request Entity Too Large” in case the request has exceeded the maximum allowed limit. A custom HTTP header MAY be used to indicate the maximum size of the request.	
[RSG-110]	If a Web API supports preference handling, it SHOULD be implemented according to IETF RFC 7240, i.e., the request HTTP header Prefer SHOULD be used and the response HTTP header Preference-Applied SHOULD be returned (echoing the original request).	
[RSG-111]	If a Web API supports preference handling, the nomenclature of preferences that MAY be set by using the Prefer header MUST be recorded in the Service Contract.	
[RSG-112]	If a Web API supports localized data, the request HTTP header Accept-Language MUST be supported to indicate the set of natural languages that are preferred in the response as specified in IETF RFC 7231.	
[RSG-113]	If the API supports long-running operations, they SHOULD be asynchronous. The following approach SHOULD be followed: The service consumer activates the service operation. The service operation returns the status code “202 Accepted” according to IETF RFC 7231 (section 6.3.3), i.e., the request has been accepted for processing but the processing has not been completed. The location of the queued task that was created is also returned with the HTTP header Location. The service consumer calls the returned Location to learn if the resource is available. If the resource is not available, the response SHOULD have the status code “200 OK”, contain the task status (for example pending) and MAY contain other information (for example, a link to cancel or delete the task using the DELETE HTTP method). If the resource is available, the response SHOULD have the status code “303 See Other” and the HTTP header Location SHOULD contain the URL to retrieve the task results.	

Rule ID	Rule description	Cross reference
[RSG-114]	Confidentiality**:** APIs and API Information MUST be identified, classified, and protected against unauthorized access, disclosure and eavesdropping at all times. The least privilege, need to know and need to share[17] principles MUST be followed.	
[RSG-115]	Integrity-Assurance: APIs and API Information MUST be protected against unauthorized modification, duplication, corruption and destruction. Information MUST be modified through approved transactions and interfaces. Systems MUST be updated using approved configuration management, change management and patch management processes.	
[RSG-116]	Availability: APIs and API Information MUST be available to authorized users at the right time as defined in the Service Level Agreements (SLAs), access-control policies and defined business processes.	
[RSG-117]	Non-repudiation: Every transaction processed or action performed by APIs MUST enforce non-repudiation through the implementation of proper auditing, authorization, authentication, and the implementation of secure paths and non-repudiation services and mechanisms.	
[RSG-118]	Authentication, Authorization, Auditing: Users, systems, APIs or devices involved in critical transactions or actions MUST be authenticated, authorized using role-based or attribute based access-control services and maintain segregation of duty. In addition, all actions MUST be logged and the authentication's strength must increase with the associated information risk.	
[RSG-119]	While developing APIs, threats, malicious use cases, secure coding techniques, transport layer security and security testing MUST be carefully considered, especially: – PUTs and POSTs – i.e.,: which change to internal data could potentially be used to attack or misinform. – DELETES – i.e.,: could be used to remove the contents of an internal resource repository – Whitelist allowable methods- to ensure that allowable HTTP Methods are properly restricted while others would return a proper response code. – Well known attacks should be considered during the threat-modeling phase of the design process to ensure that the threat risk does not increase. The threats and mitigation defined within OWASP Top Ten Cheat Sheet MUST be taken into consideration.	
[RSG-120]	While developing APIs, the standards and best practices listed below SHOULD be followed: – Secure coding best practices: OWASP Secure Coding Principles – Rest API security: REST Security Cheat Sheet – Escape inputs and cross site scripting protection: OWASP XSS Cheat Sheet – SQL Injection prevention: OWASP SQL Injection Cheat Sheet, OWASP Parameterization Cheat Sheet – Transport layer security: OWASP Transport Layer Protection Cheat Sheet	
[RSG-121]	Security testing and vulnerability assessment MUST be carried out to ensure that APIs are secure and threat-resistant. This requirement MAY be achieved by leveraging Static and Dynamic Application Security Testing (SAST/DAST), automated vulnerability management tools and penetration testing.	

Rule ID	Rule description	Cross reference
[RSG-122]	Protected services MUST only provide HTTPS endpoints. TLS 1.2, or higher, with a cipher suite that includes ECDHE for key exchange.	
[RSG-123]	When considering authentication protocols, perfect forward secrecy SHOULD be used to provide transport security. The use of insecure cryptographic algorithms and backwards compatibility to SSL 3 and TLS 1.0/1.1 SHOULD NOT be allowed.	
[RSG-124]	For maximum security and trust, a site-to-site IPSEC VPN SHOULD be established to further protect the information transmitted over insecure networks.	
[RSG-125]	The consuming application SHOULD validate the TLS certificate chain when making requests to protected resources, including checking the certificate revocation list.	
[RSG-126]	Protected services SHOULD only use valid certificates issued by a trusted certificate authority (CA).	
[RSG-127]	Tokens SHOULD be signed using secure signing algorithms that are compliant with the digital signature standard (DSS) FIPS –186-4. The RSA digital signature algorithm or the ECDSA algorithm SHOULD be considered.	
[RSG-128]	Anonymous authentication MUST only be used when the customers and the application they are using accesses information or feature with a low sensitivity level which should not require authentication, such as, public information.	
[RSG-129]	Username and password or password hash authentication MUST NOT be allowed.	
[RSG-130]	If a service is protected, then Open ID Connect SHOULD be used.	
[RSG-131]	For use of JSON Web Tokens (JWT) consider the following: – A JWT secret MUST possess high entropy to increase the work factor of a brute force attack. – Token TTL and RTTL SHOULD be as short as possible. – Sensitive information SHOULD not be stored in the JWT payload. – [RSG-130] In POST/PUT requests, sensitive data SHOULD be transferred in the request body or by request headers. – [RSG-131] In GET requests, sensitive data SHOULD be transferred in an HTTP Header. – [RSG-132] In order to minimize latency and reduce coupling between protected services, the access control decision SHOULD be taken locally by REST endpoints.	
[RSG-132]	In POST/PUT requests, sensitive data SHOULD be transferred in the request body or by request headers.	
[RSG-133]	In GET requests, sensitive data SHOULD be transferred in an HTTP Header.	
[RSG-134]	In order to minimize latency and reduce coupling between protected services, the access control decision SHOULD be taken locally by REST endpoints.	
[RSG-135]	API Keys SHOULD be used for protected and public services to prevent overwhelming their service provider with multiple requests (denial-of-service attacks). For protected services API Keys MAY be used for monetization (purchased plans), usage policy enforcement (QoS) and monitoring.	
[RSG-137]	The service provider SHOULD return along with HTTP response headers the current usage status. The following response data MAY be returned: – rate limit - rate limit (per minute) as set in the system;	

Rule ID	Rule description	Cross reference
	– rate limit remaining - remaining amount of requests allowed during the current time slot (-1 indicates that the limit has been exceeded); – rate limit reset - time (in seconds) remaining until the request counter will be reset.	
[RSG-138]	The service provider SHOULD return the status code “429 Too Many Requests” if requests are coming in too quickly.	
[RSG-139]	API Keys MUST be revoked if the client violates the usage agreement.	
[RSG-140]	API Keys SHOULD be transferred using custom HTTP headers. They SHOULD NOT be transferred using query parameters.	
[RSG-141]	API Keys SHOULD be randomly generated.	
[RSG-142]	For highly privileged services, two-way mutual authentication between the client and the server SHOULD use certificates to provide additional protection.	
[RSG-143]	Multi-factor authentication SHOULD be implemented to mitigate identity risks for application with a high-risk profile, a system processing very sensitive information or a privileged action.	
[RSG-144]	If the REST API is public then the HTTP header Access-Control-Allow-Origin MUST be set to ‘*’.	
[RSG-145]	If the REST API is protected then CORS SHOULD be used, if possible. Else, JSONP MAY be used as fallback but only for GET requests, for example, when the user is accessing using an old browser. Iframe SHOULD NOT be used.	
[RSJ-146]	If using instances described a schema, the Link header SHOULD be used to provide a link to a downloadable JSON schema ACCORDING TO RFC8288.	
[RSJ-147]	A Web API MUST implement at least Level 2 (Transport Native Properties) of RMM. Level 3 (Hypermedia) MAY be implemented to make the API completely discoverable.	
[RSJ-148]	For designing a custom hypermedia format the following set of attributes SHOULD be used enclosed into an attribute link: – href – the target URI – rel – the meaning of the target URI – self – the URI references the resource itself – next – the URI references the previous page (if used during pagination) – previous – the URI references the next page (if used during pagination) – arbitrary name v denotes the custom meaning of a relation.	

[يلي ذلك المرفق الثاني]

ANNEX II – REST IP VOCABULARY

The following IP Vocabulary is provided as an example for the RESTful Service Contracts. Particular IP Offices can extend it according to business need. The purpose of providing this information is to inform IP Offices of the types of requests that can be made when considering a HTTP GET or POST method.

Table 5: REST IP Vocabulary

Resource Name	Parameter Name	Parameter Type	Data Type	Description	Design Rule
/trademarks	applicationNumber	query	string	Returns the filed trademark identified by this application, which can be provided using WIPO ST.13 format.	
	text	query	string	Returns a list of trademarks which contain this word or series of words.	
	applicants	query	string	Returns a list of trademarks which are owned by the applicant/s identified by this the string.	
/patents	filingDate	query	string	Returns those patent applications which were filed at the IP Office on this particular date e.g., 2019-07-06	[CS-03]
	applicationNumber	query	string	Returns the filed patent application identified by this application number, which can be provided using WIPO ST.13 format.	
	inventors	query	string	Returns the filed patent application/s which are identified as being created by these inventors.	
	ipcs	query	string	Returns a list of the filed patent applications which are classified under this particular set of WIPO International Patent classifications e.g., A61M1/16.	
/designs	applicationNumber	query	string	Returns the filed design application identified by this application number, which can be using WIPO ST.13 format.	
	filingOffice	query	string	Returns the design applications that were filed at the IP Office, identified by the WIPO ST.3 code.	[CS-07]

The following technical query parameters defined in Table 6 should apply to all the REST API services:

Table 6: API technical parameters

Name	Type	Constraint	Format/Example	Description	Design Rule
format	string		`type/subtype; parameter=value`	Used for content-type negotiation (prefer a HTTP request header)	[RSG-20]
v	string		v% where % is a positive integer	Used for service versioning (prefer indicating version as path segment of the URL)	[RSG-65]
limit	positive integer	1000 > `limit` > 0	`limit=10`	The page size used for pagination	[RSG-73]
offset	positive integer	Default is 0	`offset=5`	The offset used for pagination	[RSG-73]

Name	Type	Constraint	Format/Example	Description	Design Rule
sort	comma-separated list of attributes	Directions ‘`asc`’/’`desc`’ are optional	`sort=key1:asc,key2:desc`	Multi-attribute sorting criterion	[RSG-74] – [RSG-76]
expand	comma-separated list of attributes		`expand=key1,key2`	Used for expanding the body of the returned content	[RSG-77]
count	boolean	Default is false	`count=true`	Returns the number of items in a collection (may be inline)	[RSG-80]
apiKey	string		`apiKey=abcdef12345`	Used to indicate a Web API Key (a HTTP header should be preferred)	[RSG-132] – [RSG-133]

ANNEX III - LIST OF SOAP WEB API NAMES

The following service names are recommended for SOAP Service Contracts. The recommended response data type according to the WIPO Standard ST.96 is also provided.

[Note: the table below includes some examples for further discussion and it will be completed with more examples in due course.]

Table 7: Example SOAP resource names

Service Name	Response Data Type	Description
`PatentsService`	`PatentPublication.xsd`	SOAP web service to manage patents.
`TrademarkApplicationsService`	`TrademarkApplication.xsd`	SOAP web service to manage trademark applications.
`DesignsService`	`Design.xsd`	SOAP web service to manage industrial designs.

ANNEX IV – RESTFUL WEB API GUIDELINES AND MODEL SERVICE CONTRACT

[Note: this set of guidelines will be completed for inclusion here in due course.]

Appendix

A model service contract following the design rules defined in this standard and based on the OAS (YAML) is provided below. An IP Office will be able to download the OAS and slightly adapt it in order to implement its own API.

– A draft OAS model contract: Service contract specification which outlines the business requirements and YAML API Specification.

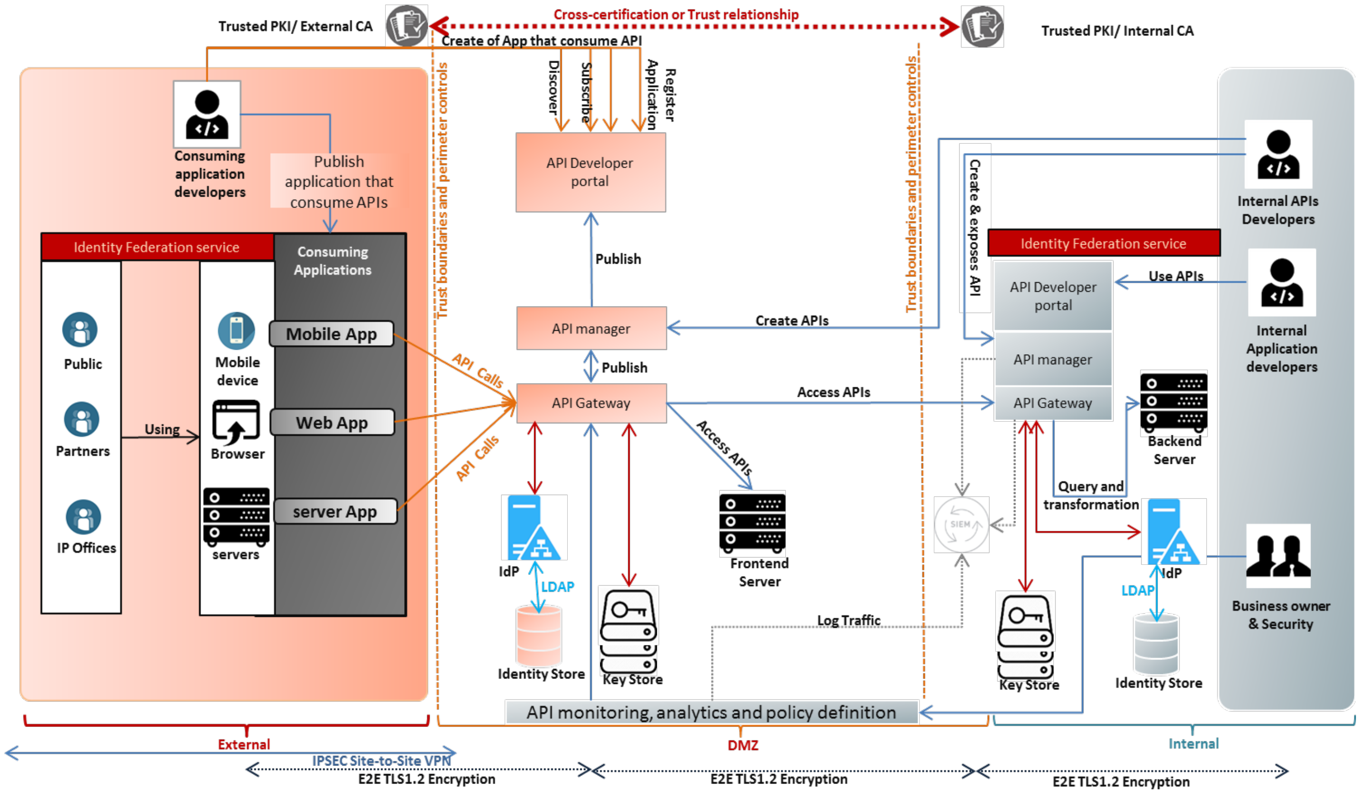
ANNEX V - SOAP WEB API MODEL SERVICE CONTRACT

A model service contract following the design rules defined in this standard and based on WSDL is provided below. An IP Office will be able to download the WSDL and slightly adapt it in order to implement its own API.

– *Note: A draft WSDL model contract will be developed and added as a separate file in due course.*

ANNEX VI – HIGH LEVEL SECURITY ARCHITECTURE BEST PRACTICES

The security architecture defines the services and mechanisms that should be implemented to enforce defined policies and rules while also providing a framework to further standardize and automate security. The core services and mechanisms of this API Security Framework (the development portal, API manager and API gateway) provide a grouping of functionality. These functions can be delivered by discrete applications, bespoke code development, via COTS products or through leveraging existing technologies that can be configured to provide these functions / services. Some of the functionality may overlap or be combined into one or more products depending on the vendor used.



The recommended security architecture SHOULD have the following API security services and mechanisms:

- A Web API portal to provide functions such as API discovery, API analytics, access to specifications and description including SLAs, social network and FAQs
- A Web API manager to provide centralized API administration and governance for API catalogues, management of registration and on-boarding of various API developer communities, API lifecycle management, application of pre-defined security profiles, and security policies lifecycle management.
- A Web API gateway to provide security automation capabilities including but not limited to centralized threat protections, centralized API authentication, authorization, logging, security policy enforcement, message encryption, monitoring, and analytics.
- A Web API monitoring and analytics service to provide functions such as advanced API services monitoring, analytics, profile usage for security baselines, changes of usage and demand.
- A credential store to provide capabilities to securely store API keys, secrets, certificates, etc.
- A trusted Certificate Authority (CA) to issue secure certificates and enable trust establishment between the various Offices.
- A Security Information and Event Management system (SIEM) to enable security logs correlation and advanced security analytics and monitoring.
- An Identity Provider to manage the identities stored in the LDAP directories and enable authentication.

ANNEX VII – HTTP STATUS CODES

It is important to align responses around the appropriate HTTP status code and to follow the standard HTTP codes. In addition to an appropriate status code, there should be a useful and concise description of the error in the body of your

HTTP response. Responses should be specific and clear so consumers can come to a conclusion very quickly when using the API.

The set of HTTP status codes is defined on the basis of in RFC7231. The status codes listed below should be used by an API, where applicable.

The following response status code categories are defined:

- 1xx: Informational - Communicates transfer protocol-level information
- 2xx: Success - Indicates that the client's request was accepted successfully
- 3xx: Redirection - Indicates that the client must take some additional action in order to complete their request
- 4xx: Client Error - This category of error status codes points the finger at clients
- 5xx: Server Error - The server takes responsibility for these error status codes

The following table consolidates the HTTP Status Codes and provides references to the relative IETF RFCs.

Table 8: HTTP Status Codes

Value	Description	Reference
100	Continue	[RFC7231, Section 6.2.1]
101	Switching Protocols	[RFC7231, Section 6.2.2]
102	Processing	[RFC2518]
103	Early Hints	[RFC8297]
104-199	Unassigned	
200	OK	[RFC7231, Section 6.3.1]
201	Created	[RFC7231, Section 6.3.2]
202	Accepted	[RFC7231, Section 6.3.3]
203	Non-Authoritative Information	[RFC7231, Section 6.3.4]
204	No Content	[RFC7231, Section 6.3.5]
205	Reset Content	[RFC7231, Section 6.3.6]
206	Partial Content	[RFC7233, Section 4.1]
207	Multi-Status	[RFC4918]
208	Already Reported	[RFC5842]
209-225	Unassigned	
226	IM Used	[RFC3229]
227-299	Unassigned	
300	Multiple Choices	[RFC7231, Section 6.4.1]
301	Moved Permanently	[RFC7231, Section 6.4.2]
302	Found	[RFC7231, Section 6.4.3]
303	See Other	[RFC7231, Section 6.4.4]
304	Not Modified	[RFC7232, Section 4.1]
305	Use Proxy	[RFC7231, Section 6.4.5]
306	(Unused)	[RFC7231, Section 6.4.6]
307	Temporary Redirect	[RFC7231, Section 6.4.7]
308	Permanent Redirect	[RFC7538]
309-399	Unassigned	
400	Bad Request	[RFC7231, Section 6.5.1]

Value	Description	Reference
401	Unauthorized	[RFC7235, Section 3.1]
402	Payment Required	[RFC7231, Section 6.5.2]
403	Forbidden	[RFC7231, Section 6.5.3]
404	Not Found	[RFC7231, Section 6.5.4]
405	Method Not Allowed	[RFC7231, Section 6.5.5]
406	Not Acceptable	[RFC7231, Section 6.5.6]
407	Proxy Authentication Required	[RFC7235, Section 3.2]
408	Request Timeout	[RFC7231, Section 6.5.7]
409	Conflict	[RFC7231, Section 6.5.8]
410	Gone	[RFC7231, Section 6.5.9]
411	Length Required	[RFC7231, Section 6.5.10]
412	Precondition Failed	[RFC7232, Section 4.2][RFC8144, Section 3.2]
413	Payload Too Large	[RFC7231, Section 6.5.11]
414	URI Too Long	[RFC7231, Section 6.5.12]
415	Unsupported Media Type	[RFC7231, Section 6.5.13][RFC7694, Section 3]
416	Range Not Satisfiable	[RFC7233, Section 4.4]
417	Expectation Failed	[RFC7231, Section 6.5.14]
418-420	Unassigned	
421	Misdirected Request	[RFC7540, Section 9.1.2]
422	Unprocessable Entity	[RFC4918]
423	Locked	[RFC4918]
424	Failed Dependency	[RFC4918]
425	Unassigned	
426	Upgrade Required	[RFC7231, Section 6.5.15]
427	Unassigned	
428	Precondition Required	[RFC6585]
429	Too Many Requests	[RFC6585]
430	Unassigned	
431	Request Header Fields Too Large	[RFC6585]
432-450	Unassigned	
451	Unavailable For Legal Reasons	[RFC7725]
452-499	Unassigned	
500	Internal Server Error	[RFC7231, Section 6.6.1]
501	Not Implemented	[RFC7231, Section 6.6.2]
502	Bad Gateway	[RFC7231, Section 6.6.3]
503	Service Unavailable	[RFC7231, Section 6.6.4]
504	Gateway Timeout	[RFC7231, Section 6.6.5]
505	HTTP Version Not Supported	[RFC7231, Section 6.6.6]
506	Variant Also Negotiates	[RFC2295]
507	Insufficient Storage	[RFC4918]
508	Loop Detected	[RFC5842]
509	Unassigned	

Value	Description	Reference
510	Not Extended	[RFC2774]
511	Network Authentication Required	[RFC6585]
512-599	Unassigned	

ANNEX VIII – REPRESENTATION TERMS

Table 9: Representation Terms

Term	Definition	Data Type
Amount	A monetary value.	Number
Category	A specifically defined division or subset in a system of classification in which all items share the same concept of taxonomy.	String
Code	A combination of one or more numbers, letters, or special characters, which is substituted for a specific meaning. Represents finite, predetermined values or free format.	String
Date	The notion of a specific point in time, expressed by year, month, and day.	String
Directory	Always preceded by PATH	String
Document	A CLOB stands for "character large object," which is a specific data type for almost all databases. Quite simply, a CLOB is a pointer to text stored outside of the table in a dedicated block. Used for XML documents. Comprised of textual information of International Trademark Registration being exchanged. XML tags identify the data items concerned with such information. TIS - Madrid development team may define the attribute XML_DOC as CLOB, pointer to Tagged Data stored outside of the table in a dedicated block.	String
Identifier	A combination of one or more integers, letters, special characters which uniquely identifies a specific instance of a business object, but which may not have a readily definable meaning.	String
Indicator	A signal of the presence, absence, or requirement of something. Recommended values are Y, N, and, “?” if needed.	Boolean
Measure	A measure is a numeric value determined by measuring an object along with the specified unit of measure. MeasureType is used to represent a kind of physical dimension such as temperature, length, speed, width, weight, volume, latitude of an object. More precisely, MeasureType should be used to measure intrinsic or physical properties of an object seen as a whole.	Number
Name	The designation of an object expressed in a word or phrase.	String
Number	A string of numeral or alphanumeric characters expressing label, value, quantity or identification.	Number, String
Percent	A number which represents a part of a whole, which will be divided by 100.	Number

Quantity	A quantity is a counted number of non-monetary units, possibly including fractions. `Quantity` is used to represent a counted number of things. `Quantity` should be used for simple properties of an object seen as a composite or collection or container to quantify or count its components. `Quantity` should always express a counted number of things, and the property will be such as total, shipped, loaded, stored. **`QuantityType`** should be used for components that require unit information; and **`xsd:nonNegativeInteger`** should be used for countable components which do not need unit information.	`Number`
Rate	A quantity or amount measured in relation to another quantity or amount.	`Number`
Text	An unformatted character string, generally in the form of words. (includes: Abbreviation, Comments.)	`String`
Time	A designation of a specified chronological point within a period.	`Date`
DateTime	The captured date and time of an event when it occurs.	`Date`
URI	The Uniform Resource Identifier that identifies where the file is located.	`String`

[يلي ذلك المرفق الثاني]

ANNEX II

118. The current DocList web service is part of the One Portal Dossier (OPD) Access services, and returns a list of documents related to a particular application number, as provided by a member of the IP5 Offices. Note that the DocList service does not return selected documents, but rather only bibliographic information including document title, document creation date and document type.

119. This specification also referred WIPO Case documentation. The current document is intended as the basis of a WIPO CASE web service, in addition to the generated OAS specification.

120. The following specification sets out the necessary requirements, for a similar web service, which will serve as an example model for the new draft Web API standard. While OPD returns responses compliant with ST.36, this updated specification will provide XML responses compliant with WIPO ST.96. If required, an ST.36 format will also be specified. The primary purpose of this document aims to provide a model example that will be incorporated in the new WIPO Web API standard. It is also intended to assist IPOs in implementing the new Standard for their web services.

RESOURCE NAME

121. Potentially the name of the resource could be:
/api/1.0/docLists

QUERY PARAMETERS

122. Table 1 indicates the parameters passed through the URI that form the request to the service.

Table 10: Query parameters

Query Parameter	Type/Format	Example	Default Value	Mandatory	Description
requesterIndividualIdentifier	String (max 4 chars)	JP	<no default value>	Yes	Individual ID at requester organization
requesterRoleName	String (String 20 chars)	OPD-System Examiner IB	<no default value>	Yes	Role at Organization: free-text
requesterOrganizationName	Enumeration: see ST.3	JP	<no default value>	Yes	ST.3 country code for requester
documentNumber	ST.13 or ST.6	AU2018244569	<no default value>	Yes	Application number requested as defined in ST.13 with mandatory IP Office code as ST.3 OR Publication number as ST.6.

Query Parameter	Type/Format	Example	Default Value	Mandatory	Description
countryCode	Enumeration: see ST.3	AU	<no default value>	No	ST.3 code indicating filing country. May be redundant as this is the first two characters of the application number.
documentKindCategory	Enumeration: see Table 6	ALL	ALL	Yes	A list of the code identifying the types of documents returned. A new type defined in *sapi* namespace.
count	True|False	True	True	No	The number of elements in a collection that can be accepted by the consumer.
sort	String	documentDate , desc	documentDate, desc	No	Multi-attribute sorting criterion for documents returned

Note: there is a mapping of these document group codes to the WIPO Case document types that is provided in the Appendix to this document.

RESPONSE

123. The high-level structure of the response is in the form:

(a) DocListsResponse identification information: OrganizationIID, OrganizationRole and IPOfficeCode (located in *sapi* namespace) – see Table 2,

(b) Bibliographic information (pat:BibliographicData) - note this specification includes only selected atomic elements were included. This list is provided in Table 2.

(c) AvailableDocumentBag (located in the sapi namespace) – see Table 3. This element is comprised of 0 to n AvailableDocument elements and is of the type AvailableDocumentBag Type.

(d) Transaction (located in the sapi namespace) – see Table 4.

124. Table 2 and Example:

```
<sapi:DocListsResponse>
     <sapi:OrganizationIID>EP</sapi:OrganizationIID>
     <sapi:OrganizationRole>EP</sapi:OrganizationRole>
     <sapi:IPOCode>EP</sapi:IPOCode>
    <pat:BibliographicData>
     …
    </pat:BibliographicData>
</sapi:DocListsResponse>
```

Table 3 provides a concordance of the proposed response body with corresponding elements of ST96, as responses from the API should be in ST.96 format. Elements that are not available in the current version of ST96 (3.1) have been created in a new namespace: www.wipo.int/standards/XMLSchema/sAPI , where *sAPI* represents the **S**tandard **API** namespace. There can then be namespaces for each of the Web services that we intend to implement created here. For instance, www.wipo.int/standards/XMLSchema/sAPI/DocList .

125. An English translation must be provided for documents supplied in the OPD web service but both the original document and the translated documents are available in the file dossier. As such, two attributes are required for the AvailableDocument element: originalLanguageCode and currentLanguageCode. Finally a third attribute, identifies the means of the translation.

Table 11: Response parameters: sapi:DocListsResponse element

Response parameter	ST.96/sapi element	Description
OrganizationIID	sapi:OrganizationIIDType	Individual ID at the organization **minOccurs=1, maxOccurs=1**
OrganizationRole	com:RoleCategoryType	Role at organization **minOccurs=1, maxOccurs=1**
IPOCode	com: IPOfficeCode	ST3 code identifying the location of the IPO **minOccurs=1, maxOccurs=1**
BibliographicData	pat:BibliographicData elements: • Application Number • Publication Number • Invention Title • Applicant • Inventor • Agent Name • WIPO No. • PCT No. • Filing Date • Application Status • Earliest Priority Date • First IPC Mark • Primary CPC mark	ST.96 response from offices which contains their implementation of this ST.96 element. Could potentially be an ST.36 response as well. **minOccurs=1, maxOccurs=1**

Example:

```
<sapi:DocListsResponse>
     <sapi:OrganizationIID>EP</sapi:OrganizationIID>
     <sapi:OrganizationRole>EP</sapi:OrganizationRole>
     <sapi:IPOCode>EP</sapi:IPOCode>
    <pat:BibliographicData>
     …
    </pat:BibliographicData>
</sapi:DocListsResponse>
```

Table 12: Response parameters – sAPI:AvailableDocument element

Response Parameter	ST.96/sapi element	Description
DocumentKindCode/DocumentKindCodeBag	sapi:DocumentKindCodeType Element (token)	See Table 6 and Table 7. Classifies type of document (as defined by OPD/WIPO Case/National Office) **minOccurs=1, maxOccurs=4**
DocumentCategory/DocumentCategoryBag	sapi:DocumentCategoryType Element (Enumeration)	The category of the document as specified by the National Offices. See Table 7. **minOccurs=0, maxOccurs=1**
NPLIndicator	sapi:NPLIndicator Element (xsd:Boolean)	A True/False Boolean flag that indicates whether this document is considered to be non-patent-literature. Implemented similar to com:ColourIndicator. **minOccurs=0, maxOccurs=1**
DocumentName	com:DocumentName Element (string)	Specify the name of the document **minOccurs=1, maxOccurs=1**
DocumentFormatCategoryBag	com:DocumentFormatCategoryType Element (Enumeration)	Specify the format of the document (eg. PDF or image formats) **minOccurs=1, maxOccurs=5**
DocumentIdentifier	sapi:DocContentIdentifier/ com:DocumentURI Element (string)	Specify the identifier to be used in DocContent Web Service Request (system identifier used to connect two web services). Could use ST.96 DocumentURI potentially. **minOccurs=1, maxOccurs=1**
DocumentDate	com:DocumentDate Element (date)	Specify the legal date of the document (YYYY-MM-DD) **minOccurs=0, maxOccurs=1**
PageTotalQuantity	com:PageTotalQuantity Element (integer)	Specify the total number of pages within the document **minOccurs=0, maxOccurs=1**
originalLanguageCode	com:ExtendedISOLanguageCodeType Attribute	Specify the original language that the document was filed MANDATORY

Response Parameter	ST.96/sapi element	Description
currentLanguageCode	com:ExtendedISOLanguageCode Type Attribute	Specify the current language of the document MANDATORY
translatorCategory	pat:TranslatorCatgeoryType Attribute	Specify how the translation was performed. e.g., Human or machine. OPTIONAL

Example:

```
<sapi:AvailableDocument com:originalLanguageCode='fr'
com:currentLanguageCode='en'  pat:translatorCategory='Human'>
     <sapi:DocumentCategoryBag>
        <sapi:DocumentCategory>AU DRAWING</sapi:DocumentCategory>
        <sapi:DocumentCategory>AU ABSTRACT</sapi:DocumentCategory>
     </sapi:DocumentCategoryBag >
     <sapi:DocumentKindCode>4</sapi:DocumentKindCode>
     <sapi:NPLIndicator>false</sapi:NPLIndicator>
     <com:DocumentName> Questions concernant la
     demande</com:DocumentName>

<com:DocumentFormatCategory>application/pdf</com:DocumentFormatCategor
y>
     <com:DocumentIdentifier>EM63EGK77322J03</com:DocumentIdentifier>
     <com:DocumentDate>2008-03-26</com:DocumentDate>
     <com:PageTotalQuantity>1</com:PageTotalQuantity>
</sapi:AvailableDocument>
```

Table 13: sapi.Transaction

Response Parameter	XML element	Description
TransactionError	com:TransactionErrorType	In case of an error, an ST.96 XML error response is returned. **minOccurs=1, maxOccurs=1**
page (limit, offset)	tuples of <limit>,<offset>	Implementation of pagination in the response. 'limit' is the number of items per page and 'offset' is the number of skipped items. i.e., pagination criteria. **minOccurs=0, maxOccurs=?**
sort	tuples of <sorting criterion>, <sorting order>	How the contents list that forms the response is sorted, i.e., sorting criteria. **minOccurs=0, maxOccurs=?**

Example:

```
<sapi:Transaction>
     <com:TransactionError>
          <com:TransactionErrorCode>200</com:TransactionErrorCode>
        <com:TransactionErrorText>OK</com:TransactionErrorText>
    </com:TransactionError>
    <page limit=”10”, offset=”0” />
    <sort by=”documentDate” order=”desc”/>
</sapi:Transaction>
```

Table 14: HTTP Status Codes (from WIPO Web API Standard). For more information on this error refer to TransactionError.

HTTP status code (from WIPO Web API Standard, Annex VII)	Description
200	Request was successful
400	Bad Request.
404	Resource not found
408	Request timeout
414	URI too large
429	Too many requests
500	Internal Server error
503	Service Unavailable

USE CASES

UC1: UP-TO-DATE LIST RETURNED (SUCCESS)

Name	UC1
User	DocList user
Goal	Return the list of documents associated with the application number [in XML] according to the origin country
Assumptions	Application number is in DOCDB format Content list is kept up-to-date by IP Office system No authentication of user – or just use SSL? Pagination is enabled
URL	http://www.wipo.int/api/1.0/docLists/au2018210291?limit=10&offset=30
Request	Query parameters within URL
Response	XML (extended ST.96)

Query parameters	Application number=2018210291
	Limit=10
	Offset=30

HTTP Verb	GET
HTTP Header	status – see Table 5

1. Requester as identified by IID and role opens up interface
2. Requester enters application number, in DOCDB format
3. Request is made by source system to destination system for document contents list
4. Contents List is received by source system and returned as a response in ST96 format
5. Up-to-date contents list of documents relating to this application are displayed to the user in desired format.

UC2: LIST NOT RETURN (ERROR)

Name	UC2
User	DocList user
Goal	Return a standard XML error
Assumptions	Application number is in DOCDB format Content list is kept up-to-date by IP Office system

Name	UC2
	Corresponds to an error code in **Error! Reference source not found.** No authentication of user or just use SSL? Pagination is enabled
URL	http://www.wipo.int/api/1.0/docLists/au2018210291?limit=10&offset=30
Request	Query parameters within URL
Response	XML (extended ST.96)

Query parameters	Application number=2018210291
	Limit=10
	Offset=30

HTTP Verb	GET
HTTP Header	Status – see Table 5

1. Requester as identified by IIF and role opens up interface
2. Requester incorrectly enters in an application number
3. Application number forms basis of the request, along with requester information
4. ST96 response is a standard error indicating that the application number cannot be found, the contents could not be retrieved, there is a delay in translation or the contents list returned is too large (greater than 1000 pages). The values for the status code and descriptions can be found in Table 5, which is a subset of Annex VII of the new WIPO Web API Standard (1).

REFERENCES

1. New WIPO Standard on Web API (Working Draft)
2. WIPO Standard for number of applications for Intellectual Property Rights
3. EPO DOCDB Reference
4. OPD Specification

APPENDIX

126. The following two tables are provided as reference material. They relate specifically to WIPO-Case and OPD implementations of the DocList web service. The Document Type Code is the code provided by the National IP Offices that identifies the type of document they are supplying to WIPO-Case. The Document Group Code is a high-level categorization provided by OPD that identifies broadly to the OPD user the type of document, indicated by color. For example, Document Group Codes 1 and 4 are colored red, as incoming documents.

127. Note, that as the document code is generated by the national offices, WIPO proposes standardization of document types, aligning them with category provided in WIPO ST.27. WIPO may also propose further detail be added to the OPD Document Group Codes.

Table 15: OPD Document Group codes[2]

Document Kind Code	Document Kind Description
1	**Application Documents**
2	**Office Actions Communications. Includes All Documents that are dispatched from office for refusal notification or determination of patent etc… - Sent to applicant, OUTGOING – Examiner communication not formality related i.e., Late Fee**

Document Kind Code	Document Kind Description
3	**Information Disclosures**
4	**Written Arguments Opinions (USPTO Office Actions) Papers received from applicants in response to the document INCOMING Include,es amendments**
5	**In-house documents including Examiner Notes/Search Results**
21	**First Office Action (substantive examination)**
22	**Intermediate Office Action (substantive examination)**
23	**Final Office Action, Decisions (substantive examination)**
101	**Documents including citations**
102	**Documents including classifications**
unknown	**No document group defined for this document.**
ALL	**Includes all the documents in the requested application**

Table 16: WIPO CASE Document Type Code- OPD Document Group Code mapping

Document Group Code	Document Type Code	IPO
1	AU DRAWING	IP AUSTRALIA
1	AU ABSTRACT	IP AUSTRALIA
4	AU AMENDMENT	IP AUSTRALIA
1	AU DESCRIPTION	IP AUSTRALIA
1	AU CLAIM	IP AUSTRALIA
1	AU CSEX	IP AUSTRALIA
1	AU SPEC	IP AUSTRALIA
	AU EXRS	IP AUSTRALIA
1	AU CORRO OUT	IP AUSTRALIA
21,101,102,2	AU A15R	IP AUSTRALIA
21,101,102,2	AU EXRP	IP AUSTRALIA
101,102	AU SIST	IP AUSTRALIA
1	CA DESCRIPTION	CIPO
1	CA CLAIMS	CIPO
1	CA ABSTRACT	CIPO
1	CA DRAWINGS	CIPO
4	CA REQUESTFORCORRECTIONTOAMENDMENT	CIPO
4	CA AMENDMENT	CIPO
4	CA REISSUE	CIPO
4	CA EXAMINATIONREINSTATEMENT	CIPO
4	CA EXTENSIONOFTIMEFOREXAMINATION	CIPO

Document Group Code	Document Type Code	IPO
4	CA AMENDMENTAFTERALLOWANCE	CIPO
4	CA PROTEST/PRIORART	CIPO
23	CA DISCLAIMER	CIPO
4	CA FINALACTION-RESPONSE	CIPO
4	CA RE-EXAMINATIONREQUESTFILED	CIPO
4	CA RESPONSETOREISSUEBOARDLETTER	CIPO
4	CA RE-EXAMINATIONREQUESTFILEDSMALLENTITYDECL	CIPO
21, 101, 102,2	CA R30(2)EXAMINERREQUISITION	CIPO
2	CA R104EXAMINERREQUISITION	CIPO
2,21,101,102	CA R29EXAMINERREQUISITION	CIPO
23,101,102,2	CA FINALACTION	CIPO
2	CA ACKNOWLEDGEMENTOFREJECTIONOFAMENDMENT	CIPO
2	CA ACKNOWLEDGEMENTOFACCEPTANCEOFAMENDMENT	CIPO
2	CA RE-EXAMINATIONREFUSED	CIPO
2	CA COMMISSIONER'SDECISION	CIPO
2	CA COMMISSIONER'SREFUSALLETTER	CIPO
2	CA R89EXAMINERREQUISITION	CIPO
2	CA R143EXAMINERREQUISITION	CIPO
2	CA PABLETTER	CIPO
101,102,2	CA INT.PRELIMINARYEXAMINATIONREPORT	CIPO
1	CA SPEC	CIPO
101,102	CA SRST	CIPO
21,101,102,2	CA EXRP	CIPO
1	CA CSEX	CIPO
1	GB DESCRIPTION	UKIPO
1	GB CLAIMS	UKIPO
21,101,2	GB EXRP-OPINION	UKIPO

Document Group Code	Document Type Code	IPO
1	GB CSEX-CLAIMS	UKIPO
1	GB SPEC-DESC	UKIPO
101,102,2	GB SRST-CORRSCH	UKIPO
101,102,2	GB SRST-AMENSCH	UKIPO
21,101,2	GB EXRP-EXAM	UKIPO
101,102,2	GB SRST-FRTHSCH	UKIPO
1,102	GB SPEC-BPUB	UKIPO
1,101,102	GB SPEC-APUB	UKIPO
1	GB CSEX-AMENCLM	UKIPO
21,101,2	GB EXRP-ABBEXAM	UKIPO
1	GB SPEC	UKIPO
1	GB CSEX	UKIPO
101,102,2	GB SRST	UKIPO
21,101,2	GB EXRP	UKIPO
21,101,102,2	GB EXRP	UKIPO
101,102	GB SRST	UKIPO
1	IL DRAWINGS	ILPO
1	IL DESCRIPTION	ILPO
1	IL CLAIMS	ILPO
2	IL SRSG	ILPO
101,102,2	IL A15R	ILPO
21,101,2	IL EXRP	ILPO
1	IL SPEC	ILPO
1	IL SEQUENCE_LISTING	ILPO
21,101,102,2	IL EXRP	ILPO
21,101,102,2	IL A15R	ILPO
101,102	IL SIST	ILPO
1	IL CSEX	ILPO
1	IL EXRS	ILPO
1	IL CORRO OUT	ILPO
1,4	IN ABSTRACT	CGPDTM
1,4	IN AFFIDAVIT	CGPDTM
4	IN AGREEMENTS	CGPDTM
4	IN AMENDMENT	CGPDTM
4	IN ANNEXURES	CGPDTM
4	IN ASSIGNMENT	CGPDTM
2,23	IN CERTIFICATES	CGPDTM
4	IN CERTIFIEDCOPY	CGPDTM
1,4	IN CLAIMS	CGPDTM
4	IN CORRESPONDENCE	CGPDTM
4	IN DECLARATION	CGPDTM
1,4	IN DESCRIPTION	CGPDTM
1,4	IN DRAWINGS	CGPDTM
4	IN EVIDENCE	CGPDTM
2,21,22,101	IN EXAMREPORTIPO	CGPDTM
2,21,22,101	IN EXRP	CGPDTM
3,4	IN FOREIGNFILINGDETAILS	CGPDTM
4	IN FORM	CGPDTM

Document Group Code	Document Type Code	IPO
2,21,22,101	IN HEARINGDOCUMENTS	CGPDTM
4	IN IPEAFORMS	CGPDTM
4	IN ISAFORMS	CGPDTM
4	IN INSPECTIONREQ	CGPDTM
4	IN ISR	CGPDTM
4	IN MARKEDCOPY	CGPDTM
4	IN NOTIFICATIONLETTERS	CGPDTM
4	IN NOTARIZEDCOPY	CGPDTM
2,23	IN OFFICEACTION	CGPDTM
4	IN OTHERDOCUMENTS	CGPDTM
4	IN OTHERREQUESTS	CGPDTM
4	IN OTHERS	CGPDTM
4	IN PA	CGPDTM
1,102	IN PCT	CGPDTM
1,4	IN PCTFORMS	CGPDTM
3,4	IN PCTPUBLICATION	CGPDTM
4	IN POA	CGPDTM
4	IN PETITION	CGPDTM
4	IN PRIORITYDOCUMENT	CGPDTM
4	IN PROOFOFRIGHT	CGPDTM
3,4	IN PROSECUTIONHISTORY	CGPDTM
4	IN REPLYTOEXAMREPORT	CGPDTM
4	IN REQFORPOSTDATING	CGPDTM
4	IN RESTOOFFICEACTIONS	CGPDTM
1	IN SEQLIST	CGPDTM
4	IN STATEMENTS	CGPDTM
4	IN TRANSLATION	CGPDTM
4	IN UNDERTAKING	CGPDTM
4	IN VERIFIEDCOPIES	CGPDTM
4	IN WITHDRAWALCORRESPONDENCE	CGPDTM
3,4	IN WRITTENSUBMISSIONS	CGPDTM
1,4	IN SPECIFICATION	CGPDTM
1	PCT A19CL	PCT
1	PCT A19LT	PCT
1	PCT A19PR	PCT
1	PCT A19PU	PCT
1	PCT ABSTR	PCT
1	PCT ACCPY	PCT
1	PCT ACSMT	PCT
1	PCT AMCLS	PCT
1	PCT APBDY	PCT
1	PCT APBNP	PCT
1	PCT APBRS	PCT
1	PCT ART19	PCT
1	PCT BSPD	PCT
1	PCT CLAIM	PCT

Document Group Code	Document Type Code	IPO
1	PCT DESCR	PCT
1	PCT DRAWI	PCT
1	PCT EAPP	PCT
1	PCT EAPS	PCT
1	PCT ETABS	PCT
1	PCT ETAPB	PCT
1	PCT ETCLM	PCT
1	PCT ETDES	PCT
1	PCT ETDRW	PCT
1	PCT ISSQ	PCT
1	PCT ISSS	PCT
1	PCT ISST	PCT
1	PCT ISSU	PCT
1	PCT ISSV	PCT
1	PCT ISSW	PCT
1	PCT PCCOM	PCT
1	PCT PDOC	PCT
1	PCT SRBTR	PCT
1	PCT TEAPP	PCT
1	PCT TEAPS	PCT
1	PCT TPDOC	PCT
2	PCT CDESR	PCT
2	PCT IP2SL	PCT
2	PCT SRSTR	PCT
2	PCT SSNEE	PCT
2	PCT TPOBS	PCT
4	PCT 3PCOR	PCT
4	PCT ACOBS	PCT
4	PCT APCOR	PCT
4	PCT APOBC	PCT
4	PCT APOBS	PCT
4	PCT WOSAC	PCT
1,102	PCT PAMPH	PCT
101,2	PCT SS501	PCT
101,2	PCT SSET	PCT
101,2	PCT SSTR	PCT
21,101,102,2	PCT ESR	PCT
21,101,102,2	PCT ETISR	PCT
21,101,102,2	PCT ISR	PCT
21,101,102,2	PCT ISRNO	PCT
2,101,102,2	PCT ITSR	PCT
2,101,102,3	PCT ROESR	PCT
2,101,102,4	PCT TESR	PCT
21,102,2	PCT A172A	PCT
21,102,3	PCT ETA17	PCT
21,102,4	PCT ETWOS	PCT
21,102,5	PCT WOSA	PCT
21,102,6	PCT WOSAR	PCT

Document Group Code	Document Type Code	IPO
21,102,7	PCT WOSNO	PCT
23,102,2	PCT BSEI	PCT
23,102,3	PCT BSIP	PCT
23,102,4	PCT ETIP1	PCT
23,102,5	PCT ETIP2	PCT
23,102,6	PCT ETIPE	PCT
23,102,7	PCT IPER	PCT
23,102,8	PCT IPR2R	PCT
23,102,9	PCT IPRP1	PCT
23,102,10	PCT IPRP2	PCT
23,2	PCT IPRP	PCT
102	PCT IASR	PCT
102	PCT PAMPHLET	PCT
4	JP A51	JPO
4	JP A521	JPO
4	JP A5210	JPO
4	JP A5211	JPO
4	JP A5212	JPO
4	JP A522	JPO
4	JP A523	JPO
4	JP A524	JPO
4	JP A525	JPO
4	JP A526	JPO
4	JP A527	JPO
4	JP A528	JPO
4	JP A529	JPO
4	JP A53	JPO
4	JP A55	JPO
4	JP A59	JPO
4	JP A601	JPO
4	JP A603	JPO
4	JP A621	JPO
4	JP A623	JPO
4	JP A624	JPO
4	JP A625	JPO
4	JP A626	JPO
4	JP A627	JPO
4,1	JP A63	JPO
4,1	JP A63	JPO
4,1	JP A631	JPO
4,1	JP A632	JPO
4,1	JP A633	JPO
4,1	JP A6330	JPO
4,1	JP A6331	JPO
4,1	JP A6332	JPO
4,1	JP A6333	JPO
4,1	JP A6333	JPO
4,1	JP A634	JPO

Document Group Code	Document Type Code	IPO
4,1	JP A6340	JPO
4,1	JP A6341	JPO
4,1	JP A6342	JPO
4,1	JP A6343	JPO
4	JP A635	JPO
4	JP A67	JPO
4	JP A681	JPO
4	JP A691	JPO
4	JP A711	JPO
4	JP A712	JPO
4	JP A7421	JPO
4	JP A7422	JPO
4	JP A7423	JPO
4	JP A7424	JPO
4	JP A7425	JPO
4	JP A7426	JPO
4	JP A7427	JPO
4	JP A7428	JPO
4	JP A7431	JPO
4	JP A7432	JPO
4	JP A7433	JPO
4	JP A7434	JPO
4	JP A7435	JPO
4	JP A7436	JPO
4	JP A7437	JPO
4	JP A761	JPO
4	JP A762	JPO
4	JP A764	JPO
4	JP A765	JPO
4	JP A781	JPO
4	JP A79	JPO
4	JP A791	JPO
4	JP A792	JPO
4	JP A80	JPO
4	JP A801	JPO
4	JP A81	JPO
4	JP A82	JPO
4	JP A821	JPO
4	JP A822	JPO
4	JP A831	JPO
4	JP A87	JPO
4	JP A871	JPO
4	JP A872	JPO
4	JP A881	JPO
4	JP A914	JPO
4	JP A915	JPO
4	JP A915	JPO
4	JP A916	JPO

Document Group Code	Document Type Code	IPO
4	JP IB101	JPO
4	JP IB101J	JPO
4,101,102	JP IB210	JPO
4,101,102	JP IB21J	JPO
4	JP IB304	JPO
4	JP IB305	JPO
4	JP IB306	JPO
4	JP IB307	JPO
4	JP IB310	JPO
4	JP IB317	JPO
4	JP IB318	JPO
4	JP IB31A	JPO
4	JP IB31B	JPO
4	JP IB31B1	JPO
4	JP IB31C	JPO
4	JP IB31C1	JPO
4,101,102	JP IB31E	JPO
4,101,102	JP IB31J	JPO
4	JP IB324	JPO
4	JP IB325	JPO
4	JP IB331	JPO
4	JP IB334	JPO
4	JP IB335	JPO
4,101,102	JP IB338	JPO
4	JP IB339	JPO
4	JP IB345	JPO
4	JP IB346	JPO
4,1,102	JP IB349	JPO
4,1	JP IB3491	JPO
4,1	JP IB3492	JPO
4,1	JP IB3493	JPO
4,1	JP IB3494	JPO
4,1	JP IB3495	JPO
4	JP IB350	JPO
4	JP IB369	JPO
4	JP IB373	JPO
4	JP IB3731	JPO
4	JP IB374	JPO
4	JP IB399	JPO
4	JP IB500	JPO
4	JP IB501	JPO
4	JP IB502	JPO
4	JP IBC101	JPO
4,101,102	JP IBC210	JPO
4,101,102	JP IBC21J	JPO
4	JP IBC304	JPO
4	JP IBC305	JPO
4	JP IBC306	JPO

Document Group Code	Document Type Code	IPO
4	JP IBC307	JPO
4	JP IBC310	JPO
4	JP IBC317	JPO
4	JP IBC31B	JPO
4	JP IBC31C	JPO
4,101,102	JP IBC31E	JPO
4,101,102	JP IBC31J	JPO
4	JP IBC324	JPO
4	JP IBC325	JPO
4	JP IBC331	JPO
4	JP IBC334	JPO
4,101,102	JP IBC338	JPO
4	JP IBC339	JPO
4	JP IBC345	JPO
4,102	JP IBC349	JPO
4	JP IBC350	JPO
2,21,23,101,102	JP A01	JPO
2,21,23,101,103	JP A01	JPO
2,23,101	JP A02	JPO
2	JP A031	JPO
2	JP A032	JPO
2	JP A033	JPO

[يلي ذلك الملحق]

APPENDIX

الملحق

Draft OAS API specification:
(https://www.wipo.int/edocs/mdocs/classifications/en/cws_7/cws_7_4-appendix1.zip)