ETSI TR 123 946 V18.0.0 (2025-01)

LTE;
5G;
Guidelines for CAPIF Usage
(3GPP TR 23.946 version 18.0.0 Release 18)

Reference

RTR/TSGS-0623946vi00

Keywords

5G,LTE

ETSI

650 Route des Lucioles
F-06921 Sophia Antipolis Cedex - FRANCE

Tel.: +33 4 92 94 42 00 Fax: +33 4 93 65 47 16

Siret N° 348 623 562 00017 - APE 7112B
Association à but non lucratif enregistrée à la
Sous-Préfecture de Grasse (06) N° w061004871

Important notice

The present document can be downloaded from the
ETSI Search & Browse Standards application.

The present document may be made available in electronic versions and/or in print. The content of any electronic and/or print versions of the present document shall not be modified without the prior written authorization of ETSI. In case of any existing or perceived difference in contents between such versions and/or in print, the prevailing version of an ETSI deliverable is the one made publicly available in PDF format on ETSI deliver repository.

Users should be aware that the present document may be revised or have its status changed,
this information is available in the Milestones listing.

If you find errors in the present document, please send your comments to
the relevant service listed under Committee Support Staff.

If you find a security vulnerability in the present document, please report it through our
Coordinated Vulnerability Disclosure (CVD) program.

Notice of disclaimer & limitation of liability

The information provided in the present deliverable is directed solely to professionals who have the appropriate degree of experience to understand and interpret its content in accordance with generally accepted engineering or other professional standard and applicable regulations.
No recommendation as to products and services or vendors is made or should be implied.
No representation or warranty is made that this deliverable is technically accurate or sufficient or conforms to any law and/or governmental rule and/or regulation and further, no representation or warranty is made of merchantability or fitness for any particular purpose or against infringement of intellectual property rights.
In no event shall ETSI be held liable for loss of profits or any other incidental or consequential damages.

Any software contained in this deliverable is provided "AS IS" with no warranties, express or implied, including but not limited to, the warranties of merchantability, fitness for a particular purpose and non-infringement of intellectual property rights and ETSI shall not be held liable in any event for any damages whatsoever (including, without limitation, damages for loss of profits, business interruption, loss of information, or any other pecuniary loss) arising out of or related to the use of or inability to use the software.

Copyright Notification

No part may be reproduced or utilized in any form or by any means, electronic or mechanical, including photocopying and microfilm except as authorized by written permission of ETSI.
The content of the PDF version shall not be modified without the written authorization of ETSI.
The copyright and the foregoing restriction extend to reproduction in all media.

© ETSI 2025.
All rights reserved.

Intellectual Property Rights

Essential patents

IPRs essential or potentially essential to normative deliverables may have been declared to ETSI. The declarations pertaining to these essential IPRs, if any, are publicly available for **ETSI members and non-members**, and can be found in ETSI SR 000 314: *"Intellectual Property Rights (IPRs); Essential, or potentially Essential, IPRs notified to ETSI in respect of ETSI standards"*, which is available from the ETSI Secretariat. Latest updates are available on the ETSI IPR online database.

Pursuant to the ETSI Directives including the ETSI IPR Policy, no investigation regarding the essentiality of IPRs, including IPR searches, has been carried out by ETSI. No guarantee can be given as to the existence of other IPRs not referenced in ETSI SR 000 314 (or the updates on the ETSI Web server) which are, or may be, or may become, essential to the present document.

Trademarks

The present document may include trademarks and/or tradenames which are asserted and/or registered by their owners. ETSI claims no ownership of these except for any which are indicated as being the property of ETSI, and conveys no right to use or reproduce any trademark and/or tradename. Mention of those trademarks in the present document does not constitute an endorsement by ETSI of products, services or organizations associated with those trademarks.

DECT™, **PLUGTESTS™**, **UMTS™** and the ETSI logo are trademarks of ETSI registered for the benefit of its Members. **3GPP™**, **LTE™** and **5G™** logo are trademarks of ETSI registered for the benefit of its Members and of the 3GPP Organizational Partners. **oneM2M™** logo is a trademark of ETSI registered for the benefit of its Members and of the oneM2M Partners. **GSM**® and the GSM logo are trademarks registered and owned by the GSM Association.

Legal Notice

This Technical Report (TR) has been produced by ETSI 3rd Generation Partnership Project (3GPP).

The present document may refer to technical specifications or reports using their 3GPP identities. These shall be interpreted as being references to the corresponding ETSI deliverables.

The cross reference between 3GPP and ETSI identities can be found at 3GPP to ETSI numbering cross-referencing.

Modal verbs terminology

In the present document "**should**", "**should not**", "**may**", "**need not**", "**will**", "**will not**", "**can**" and "**cannot**" are to be interpreted as described in clause 3.2 of the ETSI Drafting Rules (Verbal forms for the expression of provisions).

"**must**" and "**must not**" are **NOT** allowed in ETSI deliverables except when used in direct citation.

Contents

Intellectual Property Rights 2

Legal Notice 2

Modal verbs terminology 2

Foreword 5

Introduction 5

1 Scope 6

2 References 6

3 Definitions of terms, symbols and abbreviations 7
3.1 Terms 7
3.2 Symbols 7
3.3 Abbreviations 7

4 Overview of CAPIF 7
4.1 Introduction 7
4.2 Functional Architecture 7
4.3 Functional Entities 9
4.4 RNAA and CAPIF 9
4.5 Relationship between RNAA and OAuth 2.0 10
4.6 Overview CAPIF operations 11
4.6.1 General 11
4.6.2 Usage of CAPIF by API invoker 11
4.6.3 Usage of CAPIF by API provider 12

5 Role of Stakeholders for API Exposure 14
5.1 Stakeholders in CAPIF 14
5.2 Basic roles 14
5.2.1 CAPIF provider 14
5.2.2 Service API provider 15
5.2.3 API invoker 15
5.2.4 Resource owner 15
5.3 Mapping of stakeholders to CAPIF roles 15
5.4 Usage of CAPIF in different phases of an application business lifecycle 16

6 Exemplary Use Cases and Adoption 17
6.1 The deployment of the API Invoker as AF on the UE for RNAA 17
6.2 The deployment of the API Invoker as AF on the network for RNAA 19
6.3 GSMA Operator Platform deployment via CAPIF 21
6.4 CAPIF Vendor Extensibility 22
6.5 ETSI MEC deployment based on CAPIF 22
6.6 MEC Platform as an API Provider 23
6.6.1 Introduction 23
6.6.2 Registering the MEC Platform as an API Provider in CAPIF 23
6.6.3 Publishing an API for the MEC Platform 24
6.7 Open-source implementation of CAPIF 26

7 Summary 27

Annex A: Examples from OpenCAPIF 28

Annex B: CAPIF Vendor Extensibility 29

B.1 Include Vendor Specific information 29

B.2 Query for Vendor Specific information 29

B.3 ETSI MEC extensibility to CAPIF 29

Annex C: Open-source implementation of CAPIF 33
C.1 NEF Publishes an API 33
C.2 API consumption from an API Invoker 45
Annex D: CAPIF Test cases 56
D.1 General 56
D.2 Test Plan for CAPIF API Invoker Management 56
D.3 Test Plan for CAPIF API Provider Management 60
D.4 Test Plan for CAPIF API Publish Service 65
D.5 Test Plan for CAPIF Discover Service API 74
D.6 Test Plan for CAPIF API Security Service 79
D.7 Test Plan for CAPIF API Access Control Policy Service 96
D.8 Test Plan for CAPIF API Logging Service 107
D.9 Test Plan for CAPIF API Events Service 110
D.10 Test Plan for CAPIF API Auditing Service 121
Annex E: Change history 125
History 126

Foreword

This Technical Report has been produced by the 3rd Generation Partnership Project (3GPP).

The contents of the present document are subject to continuing work within the TSG and may change following formal TSG approval. Should the TSG modify the contents of the present document, it will be re-released by the TSG with an identifying change of release date and an increase in version number as follows:

Version x.y.z

where:

x the first digit:

1 presented to TSG for information;

2 presented to TSG for approval;

3 or greater indicates TSG approved document under change control.

y the second digit is incremented for all changes of substance, i.e. technical enhancements, corrections, updates, etc.

z the third digit is incremented when editorial only changes have been incorporated in the document.

Introduction

This TR provides guidelines for CAPIF usage for the benefit of the Application developer and API provider communities.

1 Scope

This TR provides guidelines for CAPIF usage for the benefit of the Application developer and API provider communities. This document also describes the usage and deployment options in CAPIF.

The work takes into consideration the work done for CAPIF in 3GPP TS 23.222 [2] and 3GPP TS 29.222 [3] and 3GPP TS 33.122 [4], and other related work outside 3GPP.

2 References

The following documents contain provisions which, through reference in this text, constitute provisions of the present document.

- References are either specific (identified by date of publication, edition number, version number, etc.) or non-specific.
- For a specific reference, subsequent revisions do not apply.
- For a non-specific reference, the latest version applies. In the case of a reference to a 3GPP document (including a GSM document), a non-specific reference implicitly refers to the latest version of that document *in the same Release as the present document*.

[1] 3GPP TR 21.905: "Vocabulary for 3GPP Specifications".

[2] 3GPP TS 23.222: "Common API Framework for 3GPP Northbound APIs; Stage 2".

[3] 3GPP TS 29.222: "Common API Framework for 3GPP Northbound APIs; Stage 3".

[4] 3GPP TS 33.122: "Security Aspects of Common API Framework for 3GPP Northbound APIs".

[5] ETSI GS MEC 011 (V3.3.1), "Multi-access Edge Computing (MEC); Edge Platform Application Enablement".

[6] 3GPP TR 23.958: "Edge Application Standards in 3GPP and Alignment with External Organizations".

[7] EVOLVED-5G ICT-41 project, "EVOLVED-5G", https://evolved-5g.eu.

[8] EVOLVED-5G NEF emulator (V2.2.3), "EVOLVED-5G NEF emulator", https://github.com/EVOLVED-5G/NEF_emulator.

[9] ETSI Software Development Group OpenCAPIF, "OpenCAPIF", https://ocf.etsi.org.

[10] ETSI Labs OCF capif, "capif", https://labs.etsi.org/rep/ocf/capif.

[11] GSMA PRD OPG.02 - "Operator Platform Telco Edge Requirements Version 7.0", https://www.gsma.com/solutions-and-impact/technologies/networks/wp-content/uploads/2024/09/OPG.02-v7.0-Operator-Platform-Requirements-and-Architecture.pdf.

[12] OpenCAPIF ETSI SDG OCF Documentation – Postman, "ETSI SDG OCF Documentation - Postman", https://ocf.etsi.org/documentation/develop/testing/postman/.

[13] OpenCAPIF ETSI SDG OCF Documentation - Common Operations, "ETSI SDG OCF Documentation - Common Operations", https://ocf.etsi.org/documentation/develop/testing/testplan/common_operations/.

[14] 3GPP TS 29.122: "T8 reference point for Northbound APIs".

[15] IETF RFC 6749 (October 2012): "The OAuth 2.0 Authorization Framework".

[16]	3GPP TS 29.500: "5G System; Technical Realization of Service Based Architecture; Stage 3".

3 Definitions of terms, symbols and abbreviations

3.1 Terms

For the purposes of the present document, the terms given in 3GPP TR 21.905 [1] and the following apply. A term defined in the present document takes precedence over the definition of the same term, if any, in 3GPP TR 21.905 [1].

For the purposes of the present document, the terms given in clause 3 of 3GPP TS 23.222 [2] and clause 3 of 3GPP TS 29.222 [3] shall also apply.

3.2 Symbols

For the purposes of the present document, the following symbols apply:

<symbol>	<Explanation>

3.3 Abbreviations

For the purposes of the present document, the abbreviations given in 3GPP TR 21.905 [1] and the following apply. An abbreviation defined in the present document takes precedence over the definition of the same abbreviation, if any, in 3GPP TR 21.905 [1].

For the purposes of the present document, the abbreviations given in clause 3 of 3GPP TS 23.222 [2] and clause 3 of 3GPP TS 29.222 [3] shall also apply.

4 Overview of CAPIF

4.1 Introduction

3GPP specifies multiple northbound API-related specifications. To avoid duplication and inconsistency of approaches between different API specifications and to specify common services (e.g. authorization), 3GPP developed a common API framework (CAPIF) that includes aspects applicable to any northbound service APIs. The key CAPIF specifications are:

- 3GPP TS 23.222 [2] defines the CAPIF architecture and procedures;
- 3GPP TS 29.222 [3] defines the API messages and protocol for CAPIF APIs; and
- 3GPP TS 33.122 [4] defines CAPIF security procedures.

4.2 Functional Architecture

The CAPIF functional architecture is specified in 3GPP TS 23.222 [2] and consists of CAPIF core function, API Provider Domain Functions and API invoker. CAPIF architecture is based on the well-known Service Oriented Architecture (SOA) design paradigm, where a Service producer (i.e. API provider) is able to publish (1) the offered service APIs which can be discovered (2) by the Service consumers (e.g. API invokers) and further can invoke (3) the discovered service APIs as per the authorization.

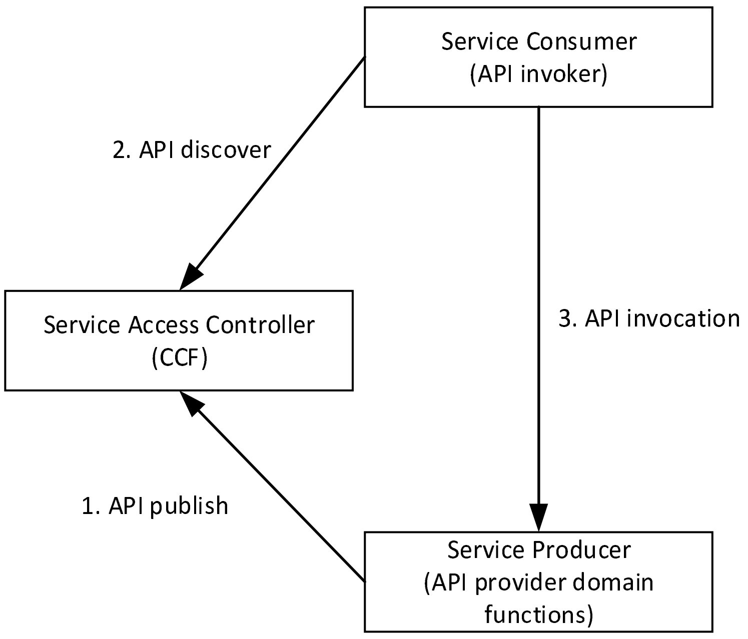
Service Consumer (API invoker)
2. API discover
Service Access Controller (CCF)
3. API invocation
1. API publish
Service Producer (API provider domain functions)

Figure 4.2-1: CAPIF based on SOA

Figure 4.2-1 provides an illustration of CAPIF based on SOA with the following relationship:

1. The functionalities related to Service Access Controller are supported by CAPIF core function.
2. The Service producer is the API Provider Domain Functions.
3. The Service consumer is the API Invoker.

Most CAPIF APIs are provided by a CAPIF core function entity and can be consumed by applications (API invokers) and application providers (CAPIF defines an API provider as three functions: API exposure, API publishing, and API management). CAPIF core function APIs enable onboarding, publishing, discovery, authentication, registration, authorization, logging, charging, monitoring, configuration, topology hiding, and other procedures. One CAPIF API (AEF security) is provided by the API exposure function.

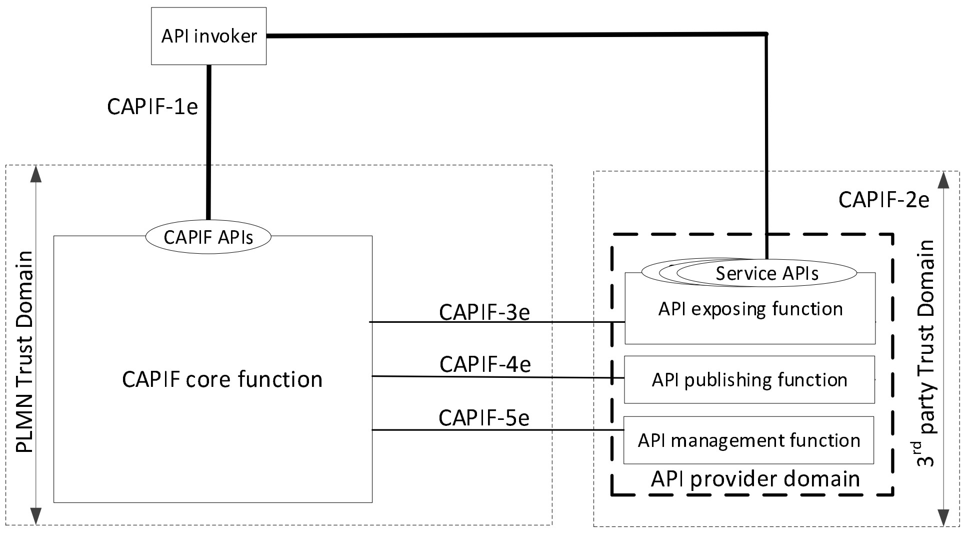

Figure 4.2-2: Functional model for the CAPIF to support 3rd party API providers

(adapted from 3GPP TS 23.222 [2] Figure 6.2.1-1: Functional model for the CAPIF to support 3rd party API providers).

The CAPIF architecture includes multiple deployment models e.g. centralized vs. distributed, single vs. multiple API providers (see 3GPP TS23.222 [2] clause 7).

4.3 Functional Entities

API invoker: Typically provided by a 3rd party application provider who has service agreement with the PLMN operator.

The API invoker supports several CAPIF capabilities such as supporting authentication, obtaining authorization, and discovering service APIs and invoking service APIs.

CAPIF core function (CCF): Supports capabilities used by other functional entities, for example the following:

- For the API invoker: authenticating the API invoker, providing authorization information, and service API discovery.
- For the API exposing function: (AEF) providing the service API access policy, providing API routing information, and charging of service API invocations.
- For the API publishing function: publishing and storing the service APIs information.
- For the API management function: providing the service API invocation log for auditing, storing configurations of the API provider policies, and updating registration information of API provider domain functions.

A CAPIF core function can also interact with another CAPIF core function for API publishing and discovery.

API exposing function (AEF): The API exposing function (AEF) is the provider of the Service APIs and is also the service communication entry point of the Service API to API invokers.

API publishing function (APF): The API publishing function (APF) enables the API provider to publish the Service APIs information to enable the discovery of Service APIs by the API invoker.

API management function (AMF): The API management function (AMF) enables the API provider to manage service APIs such as querying the Service API invocation log for auditing, monitoring the events, and configuring the API provider policies.

Authorization function: The authorization function enables receiving authorization from the resourse owner function and providing the API invoker with the authorization information which is needed to access the resource owner’s resources. This function is used in RNAA scenarios.

Resource owner function (ROF): The resource owner function enables the authorization for resource access and managing and revoking authorization for resource access. This function is used in RNAA scenarios.

4.4 RNAA and CAPIF

RNAA (Resource owner-aware Northbound API Access) is an OAuth 2.0-based authorization mechanism for CAPIF and enables API invokers to have authorized access to resources of a resource owner provided by service APIs offered by the API exposing function.

The relationship between the RNAA and CAPIF is described in this clause. The CAPIF architecture given in clause 6.2.0 of 3GPP TS 23.222 [2] can use OAuth 2.0 token's mechanism to authorize API invokers. As per OAuth 2.0, API invoker performs the function of the client, the CAPIF core function performs the function of the authorization server, and the API exposing function performs the exposure of the protected resources. The API invoker is authorized with an authorization grant type of client credentials described in clause 6.5 of 3GPP TS 33.122 [4].

The RNAA architecture given in clause 6.2.3 of 3GPP TS 23.222 [2] supports an authorization grant type of authorization code grant. The resource owner function in the RNAA architecture has the role of the resource owner, authorizing the API invoker to invoke the API exposing function.

Figure 4.4-1 shows the architectural model for the RNAA which allows the resource owner to provide authorization to the API invocation. The authorization function and the resource owner function are used in RNAA scenarios.

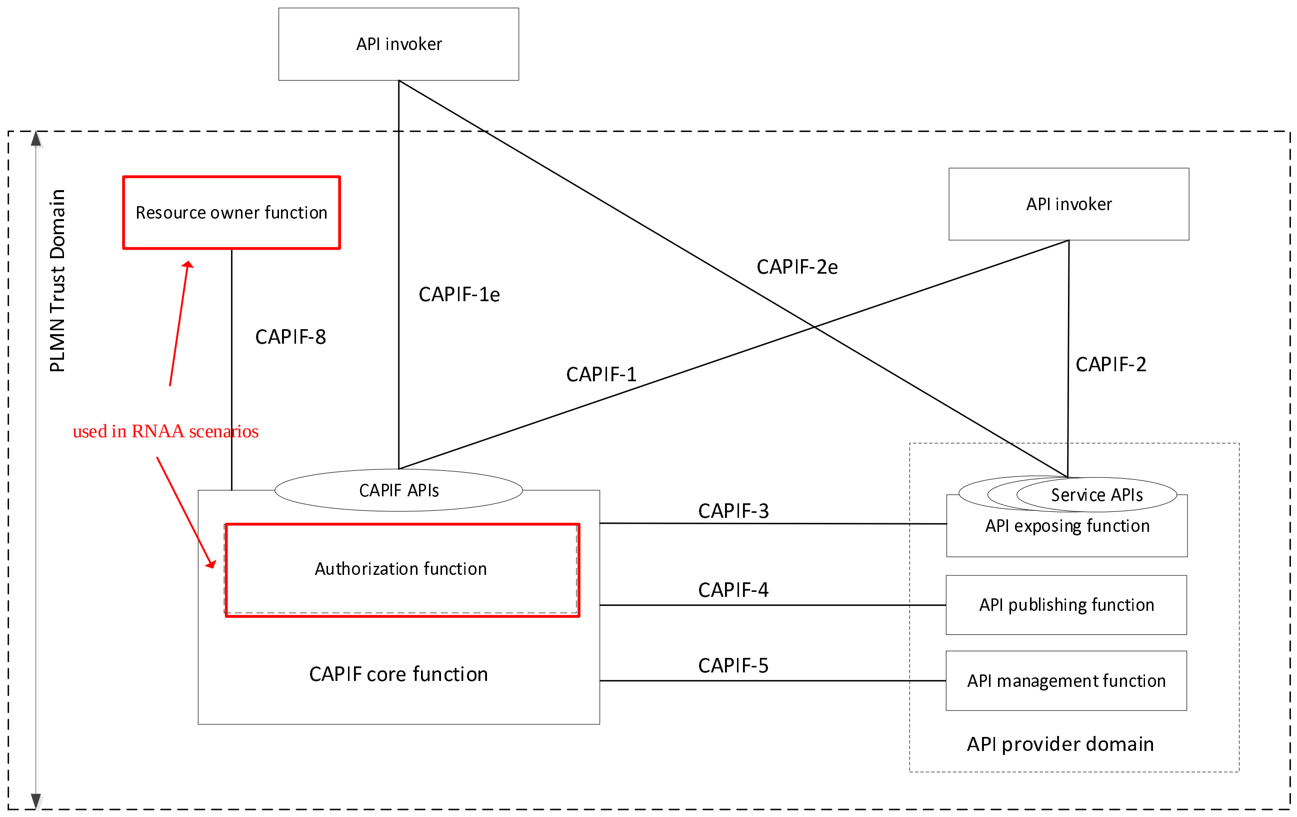

Figure 4.4-1: High level functional architecture for CAPIF supporting RNAA

4.5 Relationship between RNAA and OAuth 2.0

This clause shows the relationship between RNAA and OAuth 2.0. The Table 4.5-1 shows the mapping of OAuth 2.0 and RNAA. The details of OAuth 2.0 are specified in IETF RFC 6749 [15] and the details of RNAA are in 3GPP TS 23.222 [2].

Table 4.5-1: RNAA relationship with OAuth 2.0

Role	OAuth 2.0	RNAA
The entity capable of granting access to a protected resource.	Resource Owner	Resource owner via Resource owner function
The server hosting the protected resources, capable of accepting and responding to protected resource requests using access tokens.	Resource Server	API exposing function
The application making protected resource requests on behalf of the resource owner and with its authorization.	Client	API invoker
The server issuing access tokens to the client after successfully authenticating the resource owner and obtaining authorization.	Authorization Server	Authorization function
The software agent responsible for retrieving and facilitating end-user interaction for authorization.	User Agent	Resource owner function hosting environment

4.6 Overview CAPIF operations

4.6.1 General

This clause provides the overview of CAPIF operations from the perspective of an Application (API invoker) and API provider. The detail of the CAPIF overall operations is shown in Annex A of 3GPP TS 23.222 [2].

4.6.2 Usage of CAPIF by API invoker

Figure 4.6.2-1 describes the usage of CAPIF by API invoker for APIs which does not involve resource owner authorization.

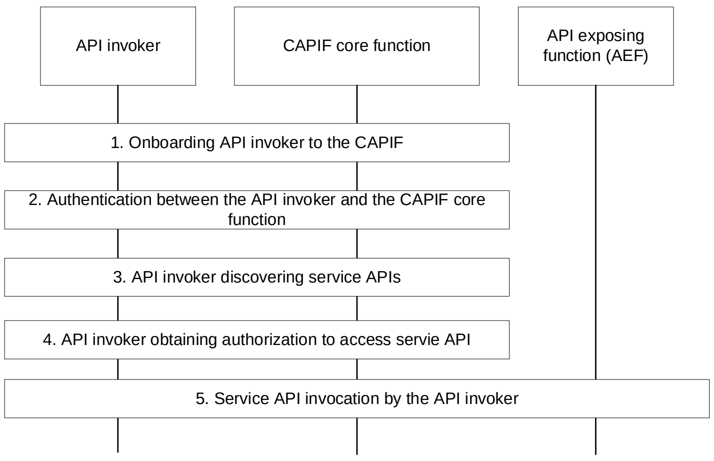
API invoker
CAPIF core function
API exposing function (AEF)
1. Onboarding API invoker to the CAPIF
2. Authentication between the API invoker and the CAPIF core function
3. API invoker discovering service APIs
4. API invoker obtaining authorization to access servie API
5. Service API invocation by the API invoker

Figure 4.6.2-1: Usage of CAPIF by API invoker for APIs which does not involve resource owner

1. API invoker onboards to the CAPIF core function, as specified in clause 8.1 of 3GPP TS 23.222 [2].
2. The API invoker authenticates with the CAPIF core function, as specified in clause 8.10 of 3GPP TS 23.222 [2].
3. The API invoker discovers the service API, as specified in clause 8.7 of 3GPP TS 23.222 [2].
4. To access the service API, the API invoker obtains authorization with the CAPIF core function, as specified in clause 8.11 of 3GPP TS 23.222 [2].
5. The API invoker performs service API invocation, as specified in clause 8.16 of 3GPP TS 23.222 [2].

Figure 4.6.2-2 describes the usage of CAPIF by API invoker for APIs which involves resource owner authorization.

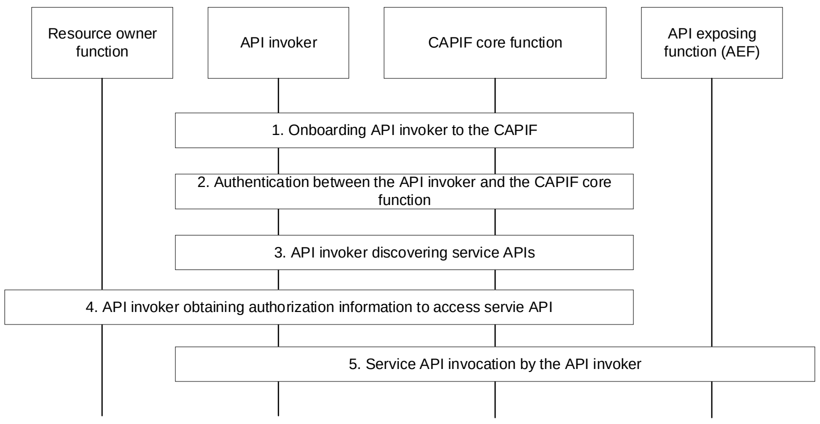
Resource owner function
API invoker
CAPIF core function
API exposing function (AEF)
1. Onboarding API invoker to the CAPIF
2. Authentication between the API invoker and the CAPIF core function
3. API invoker discovering service APIs
4. API invoker obtaining authorization information to access servie API
5. Service API invocation by the API invoker

Figure 4.6.2-2: Usage of CAPIF by API invoker for APIs which involve resource owner

1. API invoker onboards to the CAPIF core function, as specified in clause 8.1 of 3GPP TS 23.222 [2].
2. The API invoker authenticates with the CAPIF core function, as specified in clause 8.10 of 3GPP TS 23.222 [2].
3. The API invoker discovers the service API, as specified in clause 8.7 of 3GPP TS 23.222 [2].
4. To access the service API, the API invoker obtains authorization with the CAPIF core function, and then requests service API invocation as specified in clause 8.31 of 3GPP TS 23.222 [2].
5. The API invoker performs service API invocation, as specified in clause 8.16 of 3GPP TS 23.222 [2].

4.6.3 Usage of CAPIF by API provider

Figure 4.6.3-1 describes the usage of CAPIF by API provider for APIs which does not involve resource owner authorization.

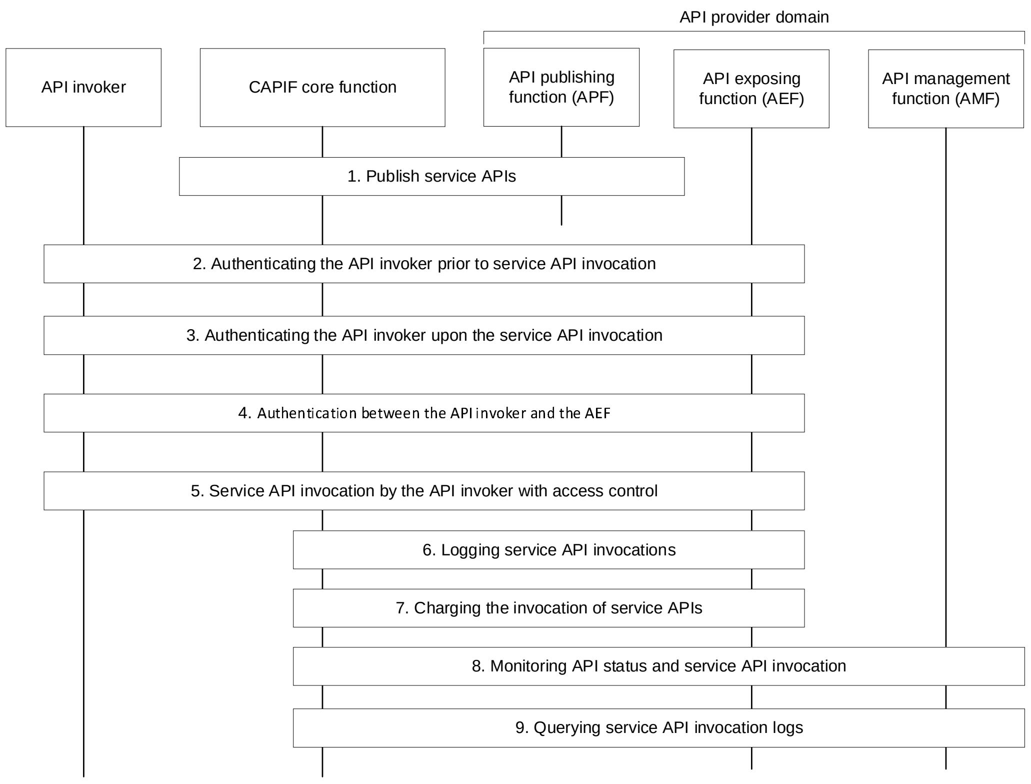
API provider domain
API invoker
CAPIF core function
API publishing function (APF)
API exposing function (AEF)
API management function (AMF)
1. Publish service APIs
2. Authenticating the API invoker prior to service API invocation
3. Authenticating the API invoker upon the service API invocation
4. Authentication between the API invoker and the AEF
5. Service API invocation by the API invoker with access control
6. Logging service API invocations
7. Charging the invocation of service APIs
8. Monitoring API status and service API invocation
9. Querying service API invocation logs

Figure 4.6.3-1: Usage of CAPIF by API provider for APIs which does not involve resource owner

1. The API publishing function sends service API publish request to the CAPIF core function, as specified in clause 8.3 of 3GPP TS 23.222 [2].
2. The API provider authenticates the API invoker prior to the service API invocation, utilizes the API exposing function, as specified in clause 8.14 of 3GPP TS 23.222 [2].
3. The API provider authenticates the API invoker upon invocation of the service APIs, utilizes the API exposing function, as specified in clause 8.15 of 3GPP TS 23.222 [2].
4. The API provider authorizes the API invoker to access, as specified in clause 8.16 of 3GPP TS 23.222 [2].
5. The API provider controls the access of the service API by the API invoker based on policy or usage limits, as specified in clause 8.16 and 8.17 of 3GPP TS 23.222 [2].
6. The API provider maintains the log of the API invocations at the CAPIF core function for services such as charging, invocation history, utilizes the API exposing function, as specified in clause 8.19 of 3GPP TS 23.222 [2].

7. The API provider facilitates charging of the API invocations, utilizes the API exposing function, as specified in clause 8.20 of 3GPP TS 23.222 [2].

8. The API provider facilitates monitoring such as API invoker's ID and IP address, utilizes the API management function, as specified in clause 8.21 of 3GPP TS 23.222 [2].

9. The API provider utilizes the API management function for auditing, as specified in clause 8.22 of 3GPP TS 23.222 [2].

Figure 4.6.3-2 describes the usage of CAPIF by API provider for APIs which involves resource owner authorization.

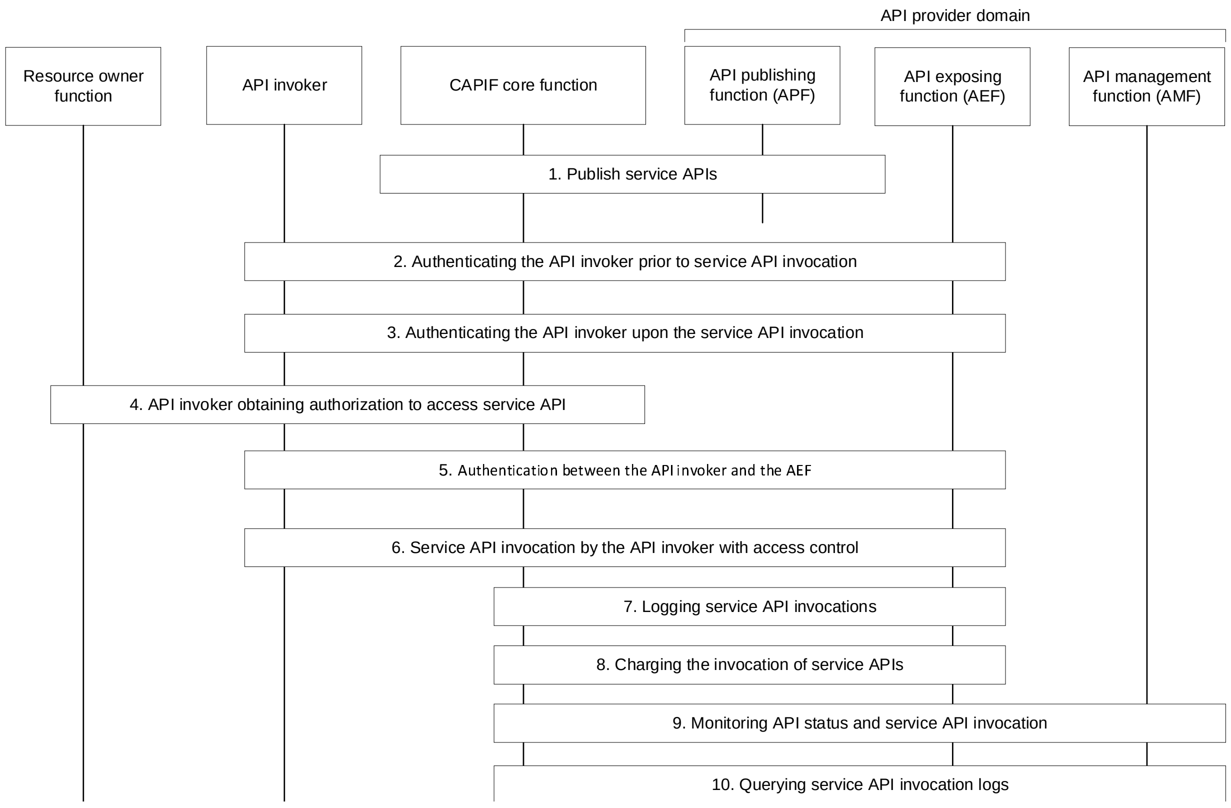
API provider domain
Resource owner function
API invoker
CAPIF core function
API publishing function (APF)
API exposing function (AEF)
API management function (AMF)
1. Publish service APIs
2. Authenticating the API invoker prior to service API invocation
3. Authenticating the API invoker upon the service API invocation
4. API invoker obtaining authorization to access service API
5. Authentication between the API invoker and the AEF
6. Service API invocation by the API invoker with access control
7. Logging service API invocations
8. Charging the invocation of service APIs
9. Monitoring API status and service API invocation
10. Querying service API invocation logs

Figure 4.6.3-2: Usage of CAPIF by API provider for APIs which involve resource owner

1. The API publishing function sends service API publish request to the CAPIF core function, as specified in clause 8.3 of 3GPP TS 23.222 [2].

2. The API provider authenticates the API invoker prior to the service API invocation, utilizes the API exposing function, as specified in clause 8.14 of 3GPP TS 23.222 [2].

3. The API provider authenticates the API invoker upon invocation of the service APIs, utilizes the API exposing function, as specified in clause 8.15 of 3GPP TS 23.222 [2].

4. API authorization is based on the authorization information obtained from the resource owner function, as specified in clause 8.31 of 3GPP TS 23.222 [2].

5. The API provider authorizes the API invoker to access, as specified in clause 8.16 of 3GPP TS 23.222 [2].

6. The API provider controls the access of the service API by the API invoker based on policy or usage limits, as specified in clause 8.16 and 8.17 of 3GPP TS 23.222 [2].

7. The API provider maintains the log of the API invocations at the CAPIF core function for services such as charging, invocation history, utilizes the API exposing function, as specified in clause 8.19 of 3GPP TS 23.222 [2].

8. The API provider facilitates charging of the API invocations, utilizes the API exposing function, as specified in clause 8.20 of 3GPP TS 23.222 [2].

9. The API provider facilitates monitoring such as API invoker's ID and IP address, utilizes the API management function, as specified in clause 8.21 of 3GPP TS 23.222 [2].

10. The API provider utilizes the API management function for auditing, as specified in clause 8.22 of 3GPP TS 23.222 [2].

5 Role of Stakeholders for API Exposure

5.1 Stakeholders in CAPIF

Several stakeholders are considered to interact within the API exposure ecosystem, for instance: MNO, NPN owners, channel aggregators, application service developers, application service providers (ASP), authorization service providers, identity service providers, resource owners, API invokers.

The functional model description for CAPIF is given in clause 6.2.0 of 3GPP TS 23.222 [2]. Different business relationships are considered for CAPIF as follows:

- Basic CAPIF business relationships are given in clause 5.1 of 3GPP TS 23.222 [2].
- CAPIF business relationships for RNAA are provided in clause 5.2 of 3GPP TS 23.222 [2].
- In case of interconnection among CAPIF providers, CAPIF business relationships are given in clause 4.12 of 3GPP TS 23.222 [2].

Some exemplary business relationships are reproduced in Figure 5.1-1.

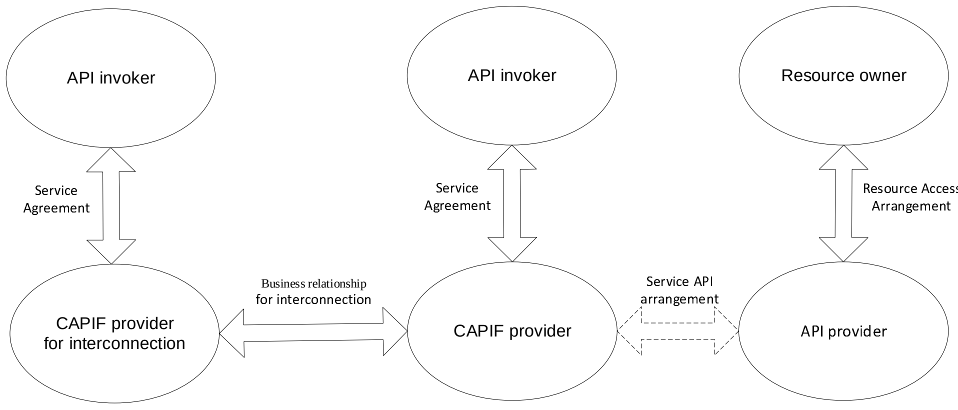

Figure 5.1-1: Basic business relationships in CAPIF

CAPIF has three main stakeholder roles (CAPIF provider, API provider, and API invoker) and RNAA has resource owner role.

5.2 Basic roles

5.2.1 CAPIF provider

The CAPIF provider contains an instance of CAPIF core function, and API provider, and API invokers and it could be a PLMN, SNPN or 3rd party. The CAPIF provider and the API provider can be part of the same organization (e.g. PLMN operator), in which case the business relationship between the two is internal to a single organization. The CAPIF provider and the API provider can be part of different organizations, in which case the business relationship between the two must exist in clause 5.1 of 3GPP TS 23.222 [2].

5.2.2 Service API provider

The service API provider hosts one or more service APIs and has a service API arrangement with CAPIF provider to offer the service APIs to the API invoker.

Requirements for enabling API invoker is given in clause 4.1.2 of 3GPP TS 23.222 [2].

5.2.3 API invoker

The API invoker is typically provided by a 3rd party application provider who has service agreement with PLMN operator and consumes the CAPIF APIs and service APIs. The API invoker may reside within the same trust domain as the PLMN operator network. There are various API invokers like application management client (used by application developers, application service provider), hosted applications (on cloud, edge or UE), and channel aggregator (who aggregates the CAPIF APIs and/or the service APIs).

Requirements for supporting 3rd party API providers is given in clause 4.1.3 of 3GPP TS 23.222 [2].

5.2.4 Resource owner

The resource owner is a UE user or an MNO subscriber capable of granting access to a protected resource related to the resource exposed by the API provider. In the current release, it is a user of the UE hosting the API invoker that can authorize the API access in clause 6.3.8 of 3GPP TS 23.222 [2].

Requirements for supporting RNAA is given in clause 4.17.2 of 3GPP TS 23.222 [2].

5.3 Mapping of stakeholders to CAPIF roles

The figure 5.3-1 provides the mapping of the stakeholders to CAPIF roles and illustrates the usage of CAPIF capabilities.

The following examples of API invoker roles are illustrated:

1. **Application management client:** The application developers utilize the CAPIF APIs using an Application management client as an API invoker to obtain the service APIs information to implement the application program. Such application programs are hosted on cloud, edge or on a UE.

2. **Hosted applications:** The Hosted applications (which are programmed to utilize the service APIs) as an API invoker invoke the CAPIF APIs and service APIs as per the application business logic.

3. **Channel Aggregator Platform:** The Channel Aggregator aggregates the APIs from one or more southbound CAPIF providers with the intention to re-expose such APIs or expose value-added APIs developed using the APIs from the southbound CAPIF providers to the northbound side Application management client or Hosted applications. The Channel Aggregator Platform as API invoker invokes the CAPIF APIs and service APIs as per its business logic.

4. **BSS/OSS System:** The BSS/OSS system of MNO enables the business relationship with the Application Service Providers (the consumers of the service APIs exposed by the MNO's exposure platform). The BSS/OSS system as an API invoker invoke the CAPIF APIs as per its business logic.

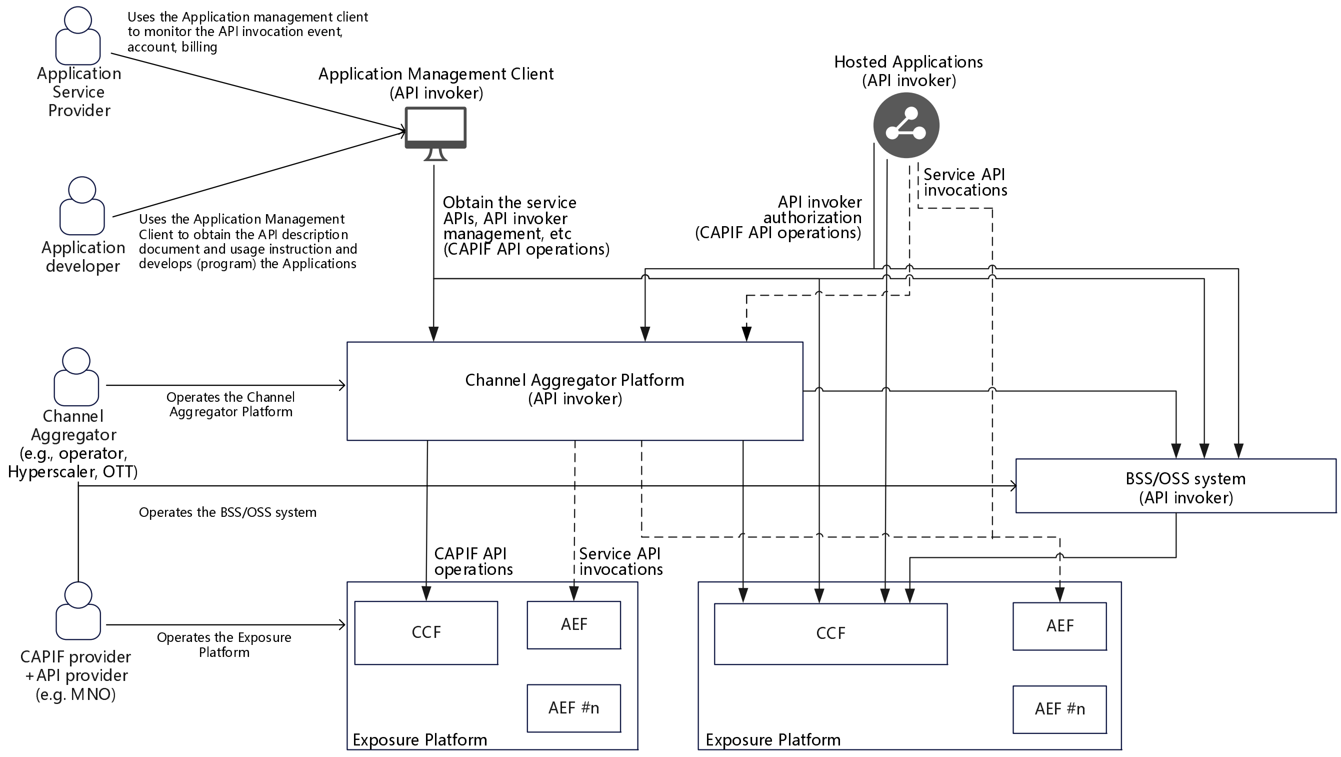
Uses the Application management client to monitor the API invocation event, account, billing
Application Service Provider
Application Management Client (API invoker)
Hosted Applications (API invoker)
Service API invocations
Application developer
Uses the Application Management Client to obtain the API description document and usage instruction and develops (program) the Applications
Obtain the service APIs, API invoker management, etc (CAPIF API operations)
API invoker authorization (CAPIF API operations)
Channel Aggregator (e.g., operator, Hyperscaler, OTT)
Operates the Channel Aggregator Platform
Channel Aggregator Platform (API invoker)
BSS/OSS system (API invoker)
Operates the BSS/OSS system
CAPIF API operations
Service API invocations
CAPIF provider +API provider (e.g. MNO)
Operates the Exposure Platform
CCF
AEF
AEF #n
Exposure Platform
CCF
AEF
AEF #n
Exposure Platform

Figure 5.3-1: Mapping of the stakeholders to CAPIF roles

5.4 Usage of CAPIF in different phases of an application business lifecycle

This clause shows an example of CAPIF usage in application service system. In this example, there are following stakeholders/roles: application service provider, application developer, and CAPIF provider and service API provider as defined in 3GPP TS 23.222 [2].

Figure 5.4-1 shows the whole service flow of CAPIF usage by the 3rd party during the application business lifecycle. The application business lifecycle includes 3 phases:

Phase 1. Application development preparation phase: Before the application developer develops the application, the developer determines the candidate service APIs to be used for the application. The developer selects the CAPIF provider and the service API provider. Then the developer searches the interested service API(s) information, e.g. available service API, service API name, and so on via the utilizes the application management client (act as API invoker), which invokes the CAPIF_Discover_Service_API to obtain that service API information from the CCF. Then, the developer creates an account and gets authorization to access CAPIF core function. This procedure follows onboarding of API invoker as specified in clause 8.1 in 3GPP TS 23.222 [2]. The developer may find the service API provider at any time later.

Phase 2. Application development and testing phase: The application developer obtains the detailed service API information including API usage instructions which enables the application developer to program the service API into the application. In addition, the developer may also obtain the security credential of the service API before programming it (option #1). Later, the application developer can test the application which invokes the corresponding CAPIF API and service API (e.g. in a sandbox mode).

Phase 3. Application deployment and execution phase: After the development, the application is commercial released. The host application (acting as API invoker) obtains the authorization from CAPIF core function to invoke the service API as specified in clause 8.11 of 3GPP TS 23.222 [2]. The host application may obtain the security credential of the service API invocation (Option #2). Upon receiving the access token from CAPIF core function, the application initiates the service API invocation, as specified in clause 8.16 in 3GPP TS 23.222 [2]. In RNAA scenario, the API invoker obtains authorization with the CAPIF core function as specified in clause 8.31 of 3GPP TS 23.222 [2], and then requests service API invocation.

NOTE: There are two options about obtaining the security credential of the service API invocation. Option #1 is fetching it during phase 1 or phase 2 and program it into the host application. In this option, the security can not be dynamically refreshed at any time. Option #2 is fetching it during the host application is running. In this option, the security can be dynamically refreshed at any time.

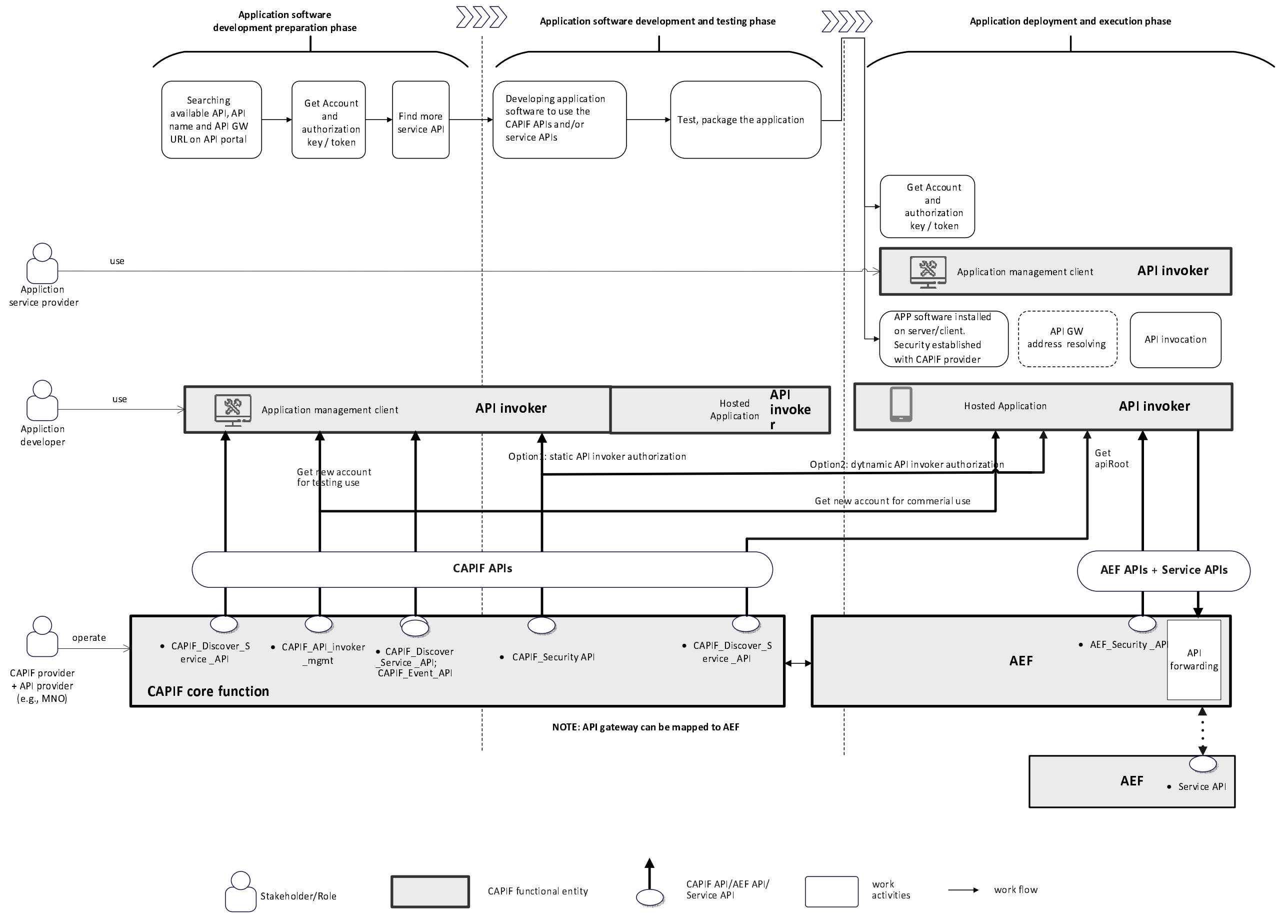
Application software development preparation phase
Application software development and testing phase
Application deployment and execution phase
Searching available API, API name and API GW URL on API portal
Get Account and authorization key / token
Find more service API
Developing application software to use the CAPIF APIs and/or service APIs
Test, package the application
Get Account and authorization key / token
Appliction service provider
use
Application management client
API invoker
APP software installed on server/client. Security established with CAPIF provider
API GW address resolving
API invocation
Appliction developer
use
Application management client
API invoker
Hosted Application
API invoke r
Hosted Application
API invoker
Option1: static API invoker authorization
Option2: dytnamic API invoker authorization
Get new account for testing use
Get new account for commerial use
Get apiRoot
CAPIF APIs
AEF APIs + Service APIs
CAPIF provider + API provider (e.g., MNO)
operate
CAPIF_Discover_Service _API
CAPIF_API_invoker _mgmt
CAPIF_Discover _Service _API; CAPIF_Event_API
CAPIF_Security API
CAPIF_Discover_Service _API
CAPIF core function
AEF
AEF_Security _API
API forwarding
NOTE: API gateway can be mapped to AEF
AEF
Service API
Stakeholder/Role
CAPIF functional entity
CAPIF API/AEF API/ Service API
work activities
work flow

Figure 5.4-1: Usage of CAPIF in different phases of an application business lifecycle

6 Exemplary Use Cases and Adoption

6.1 The deployment of the API Invoker as AF on the UE for RNAA

Several RNAA deployments are described in clause 7.5 of 3GPP TS 23.222 [2], and the following is an example of a RNAA deployment in which API invoker is deployed as AF on the UE.

In this deployment scenario, UE, which functions as both the API Invoker (AF) and the resource owner function, requests resource owner consent from the resource owner function through CAPIF core function to obtain the resource (e.g., QoS).

Figure 6.1-1 illustrates integrated fundamental deployment of the API Invoker as AF on the UE for RNAA.

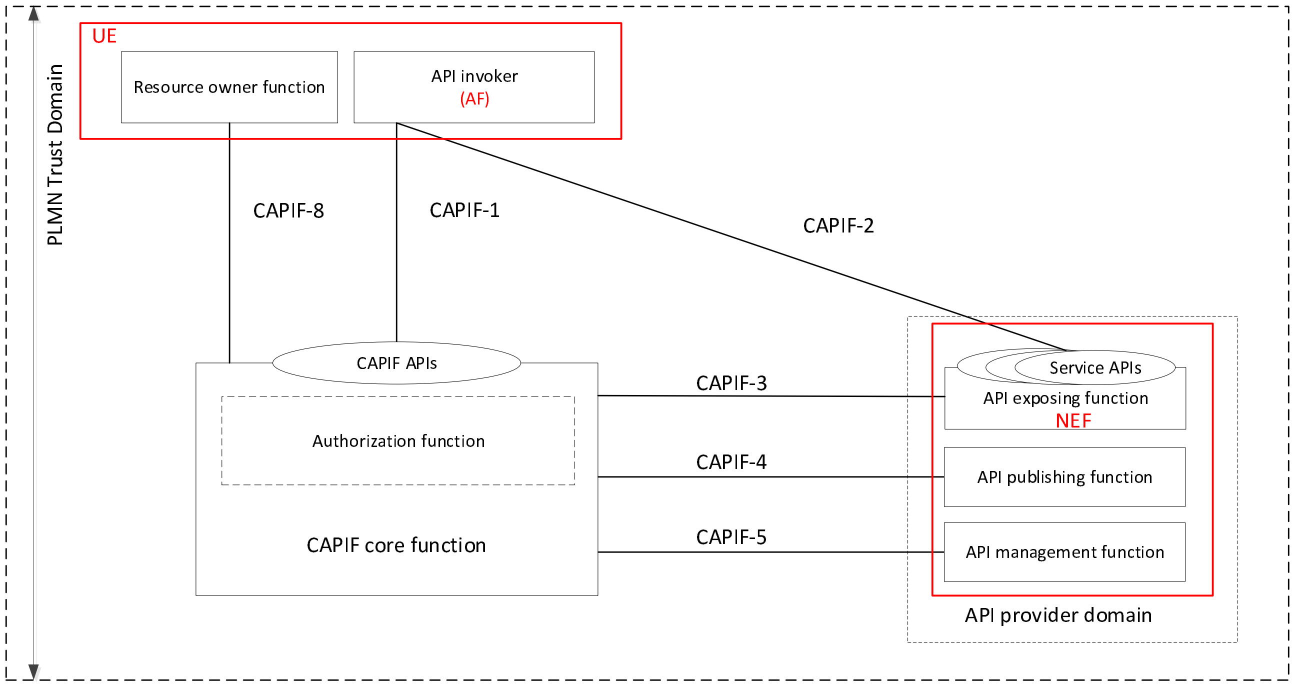

Figure 6.1-1: The deployment of the API Invoker as AF on the UE for RNAA

Figure 6.1-2 illustrates the procedure for the deployment of the API Invoker as AF on the UE for RNAA.

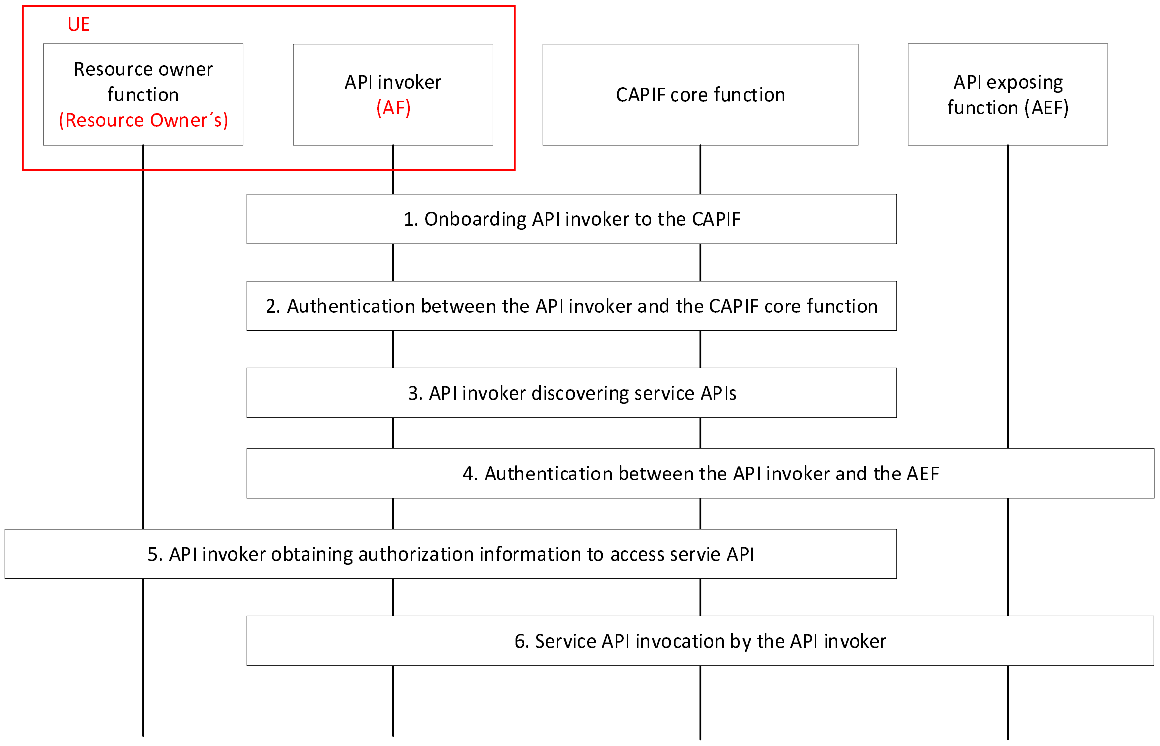
UE
Resource owner function (Resource Owner´s)
API invoker (AF)
CAPIF core function
API exposing function (AEF)
1. Onboarding API invoker to the CAPIF
2. Authentication between the API invoker and the CAPIF core function
3. API invoker discovering service APIs
4. Authentication between the API invoker and the AEF
5. API invoker obtaining authorization information to access servie API
6. Service API invocation by the API invoker

Figure 6.1-2: The deployment of the API Invoker as AF on the UE for RNAA

The following procedure is for the deployment of the API Invoker as AF on the UE for RNAA:

1. The API invoker triggers onboard API invoker request towards the CAPIF core function. The CAPIF core function then verifies the request and responds to the API invoker with the result.

2. The API invoker sends an obtain service API authorization request to the CAPIF core function for obtaining permission to access the service API. The CAPIF core function validates the authentication of the API invoker and checks whether the API invoker is permitted to access the requested service API. Then, the CAPIF core function responds to the API invoker with the authorization information required to access the service APIs.

3. The API invoker sends a service API discover request to the CAPIF core function and the CAPIF core function verifies the identity of the API invoker. Then the CAPIF core function sends a service API discover response to the API invoker with the list of service API information for which the API invoker has the required authorization.

4. The AEF sends an obtain access control policy request to the CAPIF core function for obtaining the policy to perform the access control on service API invocations. The CAPIF core function checks whether the AEF is authorized to receive the access control policy corresponding to the service APIs requested. The AEF is provided the access control policy for the service API via an obtain access control policy response.

5. The API invoker requests the resource owner's authorization information to invoke the service API, exposed by the API exposing function. The authorization function facilitates this request by involving the resource owner. Then, the API invoker sends a service API invocation request to the API exposing function, which in turn sends a response back to the API invoker.

6. The API invoker (UE) sends service API invocation request to the and the AEF checks whether the API invoker is authorized to invoke that service API, based on the authorization information. The API invoker receives the service API invocation response as a result of the service API invocation.

6.2 The deployment of the API Invoker as AF on the network for RNAA

In this deployment scenario, the UE, which functions as both the application and the resource owner function, initiates the application's requests. The API Invoker then requests resource owner consent from the resource owner function through CAPIF core function to obtain the resource (e.g., QoS).

Figure 6.6-1 illustrates integrated fundamental deployment of the API Invoker as AF on the network for RNAA.

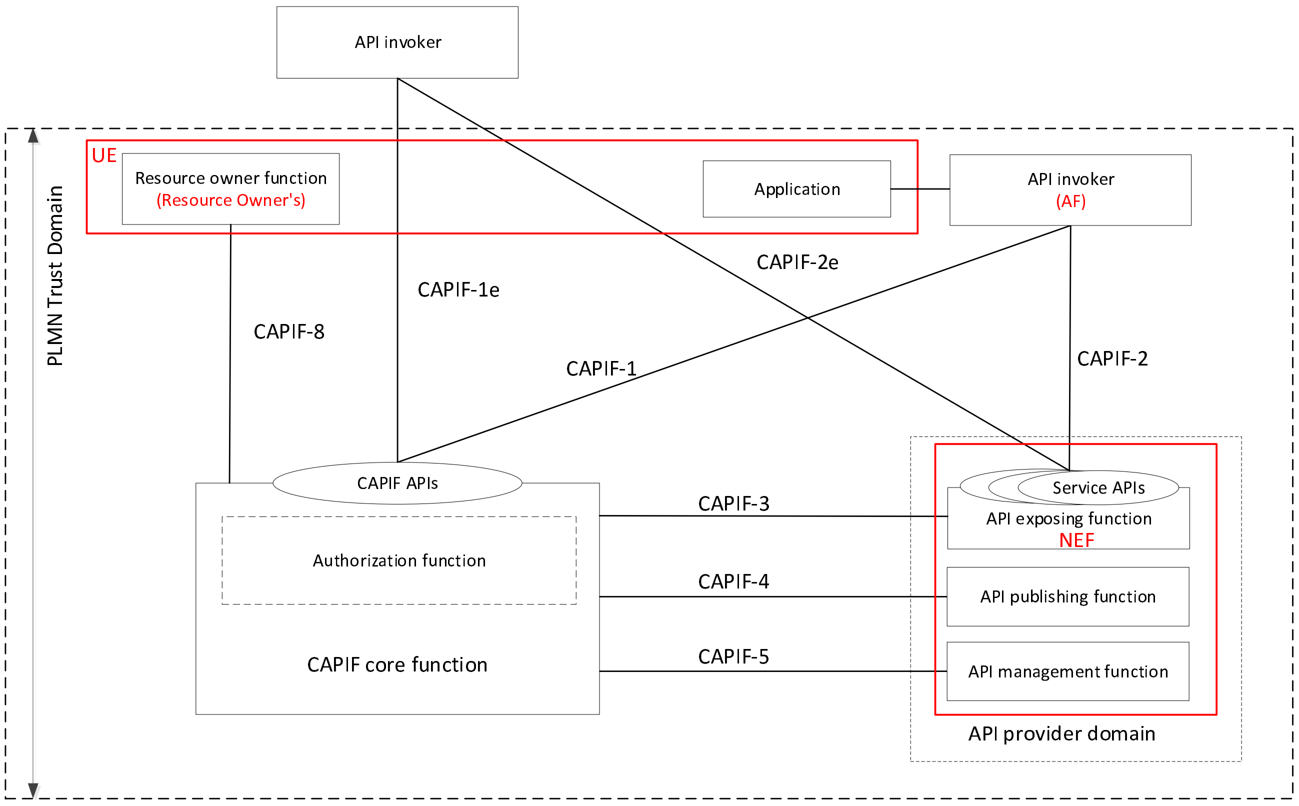

Figure 6.6-1: The deployment of the API Invoker as AF on the network for RNAA

Figure 6.6-2 illustrates the procedure for the deployment of the API Invoker as AF on the network for RNAA.

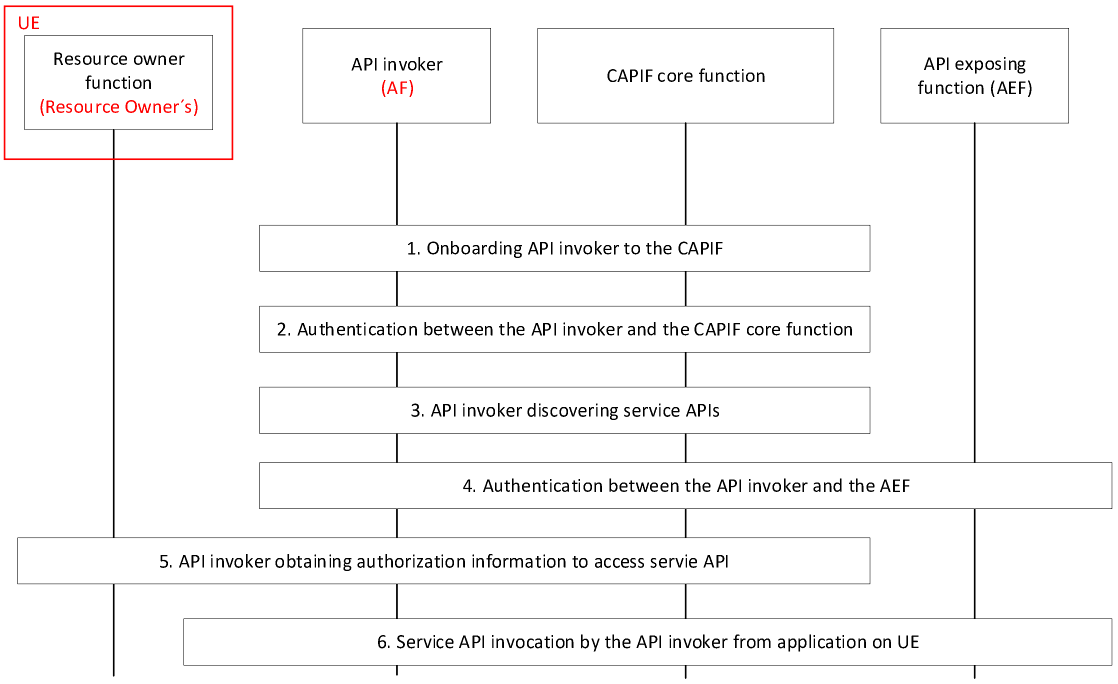
UE
Resource owner function
(Resource Owner's)
API invoker
(AF)
CAPIF core function
API exposing function (AEF)
1. Onboarding API invoker to the CAPIF
2. Authentication between the API invoker and the CAPIF core function
3. API invoker discovering service APIs
4. Authentication between the API invoker and the AEF
5. API invoker obtaining authorization information to access servie API
6. Service API invocation by the API invoker from application on UE

Figure 6.2-2: The procedure for the deployment of the API Invoker as AF on the network for RNAA

The following procedure is for API Invoker deployment on the network as AF for RNAA:

1. The API invoker triggers onboard API invoker request towards the CAPIF core function. The CAPIF core function then verifies the request and responds to the API invoker with the result.

2. The API invoker sends an obtain service API authorization request to the CAPIF core function for obtaining permission to access the service API. The CAPIF core function validates the authentication of the API invoker and checks whether the API invoker is permitted to access the requested service API. Then, the CAPIF core function responds to the API invoker with the authorization information required to access the service APIs.

3. The API invoker sends a service API discover request to the CAPIF core function and the CAPIF core function verifies the identity of the API invoker. Then the CAPIF core function sends a service API discover response to the API invoker with the list of service API information for which the API invoker has the required authorization.

4. The AEF sends an obtain access control policy request to the CAPIF core function for obtaining the policy to perform the access control on service API invocations. The CAPIF core function checks whether the AEF is authorized to receive the access control policy corresponding to the service APIs requested. The AEF is provided the access control policy for the service API via an obtain access control policy response.

5. The API exposing function obtains the resource owner consent from the resource owner. The API invoker requests the resource owner's authorization information to invoke the service API, exposed by the API exposing function. The authorization function facilitates this request by involving the resource owner. Then, the API invoker sends a service API invocation request to the API exposing function, which in turn sends a response back to the API invoker.

6. From the application's request on UE, the API invoker sends service API invocation request to the AEF and the AEF checks whether the API invoker is authorized to invoke that service API, based on the authorization information. The API invoker receives the service API invocation response as a result of the service API invocation.

6.3 GSMA Operator Platform deployment via CAPIF

The GSMA Operator Platform defines a common platform exposing operator services / capabilities to customers / application providers in the 5G-era. The requirement that an Operator Platform's SBI-NR can work with CAPIF when available is specified in clause 5.1.4.2.2 of GSMA PRD OPG.02 [11] and application session continuity support for handovers between 4G and 5G based on combined SCEF+NEF via CAPIF is described in clause 5.2.2.6.7 of GSMA PRD OPG.02 [11].

Figure 6.3-1 illustrates integrated fundamental GSMA Operator Platform deployment of the SCEF and the NEF with the CAPIF. In this deployment model, the GSMA Operator Platform functions as the API invoker.

NOTE: An Operator Platform's SBI-NR can work without CAPIF. If CAPIF is not supported, the SBI-NR API will provide an alternate means of providing these functions.

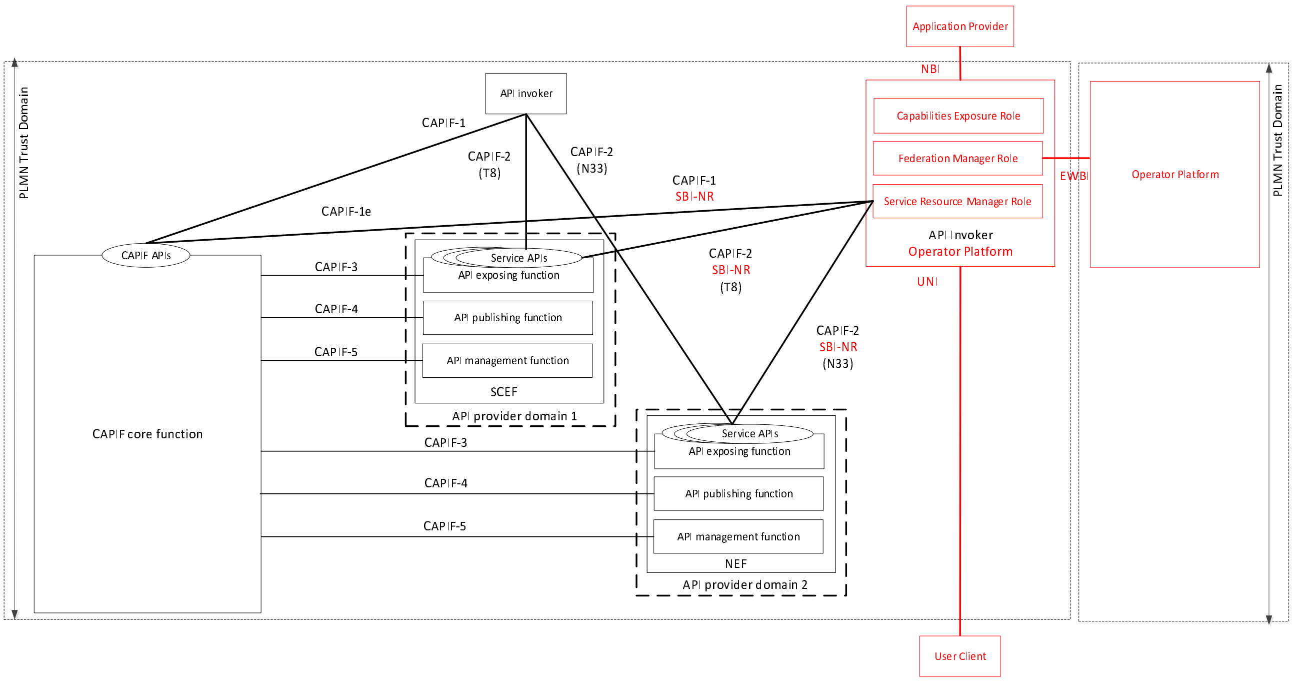

Figure 6.3-1: Integrated GSMA Operator Platform deployment of the SCEF and the NEF with the CAPIF

6.4 CAPIF Vendor Extensibility

Vendor extensibility is a CAPIF feature that enables third-party API frameworks to re-use CAPIF functionality, adapted to their requirements, by extending either CAPIF APIs or CAPIF data models. A detailed description of vendor extensibility mechanisms is provided on 3GPP TS 29.122 [14] and 3GPP TS 29.500 [16]. The CAPIF APIs supporting this feature are the following:

- CAPIF_Publish_Service_API, including “VendorExt” option in the supported feature list (see Table 8.1.6-1 of 3GPP TS 29.222 [3], which lists 3 features), indicating the support for CAPIF vendor specific extensions on the ServiceAPIDescription data model.
- CAPIF_Discover_Service_API, including “VendSpecQueryParams” option in the supported feature list (see Table 8.2.6-1 of 3GPP TS 29.222 [3], which lists 9 features), indicating the support of vendor specific query parameters in API discovery requests.

Annex B provides more details of CAPIF vendor extensibility and illustrates ETSI MEC leveraging the CAPIF vendor extensibility feature.

6.5 ETSI MEC deployment based on CAPIF

The ETSI MEC platform is the collection of essential functionalities required to run MEC applications on a particular Virtualisation infrastructure and enable them to provide and consume MEC services. In clause 9 of ETSI GS MEC 011 V3.3.1 [5], a description is provided on how the MEC service management APIs can be realized by CAPIF APIs. The clause describes how MEC deployments can reuse CAPIF functionality to harmonize between MEC and CAPIF, e.g., to integrate MEC applications into a joint 3GPP-MEC deployment. To enable such harmonization, a variant of the MEC service management functionality is defined that leverages the CAPIF Publish Service API, CAPIF Discover Service API and CAPIF Events API. The variant is referred to as the "MEC profile of CAPIF".

This deployment option for ETSI MEC based on CAPIF is also depicted at a high-level in clause 7.2 of 3GPP TR 23.958 [6].

6.6 MEC Platform as an API Provider

6.6.1 Introduction

This clause outlines the registration process for the MEC platform as an API provider within the CAPIF framework, describing the necessary steps and requirements for API publication and management.

6.6.2 Registering the MEC Platform as an API Provider in CAPIF

To register the MEC platform as an API provider in CAPIF, an access token (`access_token`) is required to authorize the provider's onboarding. Once the required information is available, the provider submits an HTTP POST request to the following URI:

`POST {apiRoot}/api-provider-management/v1/registrations`

Request Headers:

- `Authorization: Bearer {access_token}`
- `Content-Type: application/json`

In the request body, include the `regSec` field (holding the token) and provide a Certificate Signing Request (CSR) along with relevant details for each provider function: Application Management Function (AMF), API Publishing Function (APF), and API Exposing Function (AEF). Additionally, a domain name and other provider-related information can be specified.

Example Request Body:

```
HTTP POST https://capif.mec.example.com/api-provider-management/v1/registrations

Request headers:
Authorization: Bearer eyJhbGci0iJSUzIlNiIsInR5cCI6IkpXVCJ9
Content-Type: application/json

Request body:
{
  "regSec": "eyJhbGci0iJSUzIlNiIsInR5cCI6IkpXVCJ9",
  "apiProvFuncs": [
    {
      "regInfo": {
        "apiProvPubKey": "-----BEGIN CERTIFICATE REQUEST-----\nCSR Body\n-----
END CERTIFICATE REQUEST-----\n"
      },
      "apiProvFuncRole": "AEF",
      "apiProvFuncInfo": "AEF for MEC Platform"
    },
    {
      "regInfo": {
        "apiProvPubKey": "-----BEGIN CERTIFICATE REQUEST-----\nCSR Body\n-----
END CERTIFICATE REQUEST-----\n"
      },
      "apiProvFuncRole": "APF",
      "apiProvFuncInfo": "APF for MEC Platform"
    },
    {
      "regInfo": {
        "apiProvPubKey": "-----BEGIN CERTIFICATE REQUEST-----\nCSR Body\n-----
END CERTIFICATE REQUEST-----\n"
      },
      "apiProvFuncRole": "AMF",
      "apiProvFuncInfo": "AMF for MEC Platform"
    }
  ],
  "apiProvDomInfo": "MEC Platform",
  "apiProvName": "OCF_MEC"
}
```

Upon successful registration, CAPIF returns the location of the newly created resource in the Location field of the header. The response body includes signed certificates and IDs for each provider function, as well as the domain ID.

```
Response headers:
Location: https://capif.mec.example.com/api-provider-
management/v1/registrations/3a4df5f6e30a5ee34de346004ee6bb

Response body:
{
  "apiProvDomId": "3a4df5f6e30a5ee34de346004ee6bb",
  "regSec": "eyJhbGci0iJSUzIlNiIsInR5cCI6IkpXVCJ9",
  "apiProvFuncs": [
    {
      "apiProvFuncId": "AEFb1a522967511fd6d6def0cbb3b5f05",
      "regInfo": {
        "apiProvPubKey": "-----BEGIN CERTIFICATE REQUEST-----\nCSR\n-----END
CERTIFICATE REQUEST-----\n",
        "apiProvCert": "-----BEGIN CERTIFICATE -----\nAEF CERTIFICATE\n-----END
CERTIFICATE-----\n"
      },
      "apiProvFuncRole": "AEF",
      "apiProvFuncInfo": "AEF for MEC Platform "
    },
    {
      "apiProvFuncId": "APF54f49ee44f67c81fc7d7f321195531",
      "regInfo": {
        "apiProvPubKey": "-----BEGIN CERTIFICATE REQUEST-----\nCSR\n-----END
CERTIFICATE REQUEST-----\n",
        "apiProvCert": "-----BEGIN CERTIFICATE -----\nAPF CERTIFICATE\n-----END
CERTIFICATE-----\n"
      },
      "apiProvFuncRole": "APF",
      "apiProvFuncInfo": "APF for MEC Platform"
    },
    {
      "apiProvFuncId": "AMFOdbe115f6759f0fec943321dc2184d",
      "regInfo": {
        "apiProvPubKey": "-----BEGIN CERTIFICATE REQUEST-----\nCSR\n-----END
CERTIFICATE REQUEST-----\n",
        "apiProvCert": "-----BEGIN CERTIFICATE -----\nAMF CERTIFICATE\n-----END
CERTIFICATE-----\n"
      },
      "apiProvFuncRole": "AMF",
      "apiProvFuncInfo": "AMF for MEC Platform"
    }
  ],
  "apiProvDomInfo": "MEC Platform",
  "apiProvName": "OCF_MEC"
}
```

6.6.3 Publishing an API for the MEC Platform

To publish an API, the MEC platform uses mutual TLS authentication with the APF certificate to make an HTTP POST request to the following URI:

```
POST {apiRoot}/published-apis/v1/{apfId}/service-apis
```

The request body should specify details of the API to be published, including name, status, version, resources, and protocol details.

Example Request Body:

```
HTTP POST https://capif.mec.example.com/published-
apis/v1/APF54f49ee44f67c81fc7d7f321195531/service-apis

Request body:
{
  "apiName": "Application Instance Registration",
  "apiStatus": {
    "aefIds": ["AEFb1a522967511fd6d6def0cbb3b5f05"]
  },
  "aefProfiles": [
    {
      "aefId": "AEFb1a522967511fd6d6def0cbb3b5f05",
      "versions": [
        {
          "apiVersion": "v1",
          "expiry": "2100-11-30T10:32:02.004Z",
          "resources": [
            {
              "resourceName": "MEC RESOURCE NAME",
              "commType": "REQUEST_RESPONSE",
              "uri": "/mec_app_support/v2/registrations",
              "custOpName": "http_post",
              "operations": [
                "POST"
              ],
              "description": "App Registration Resource"
            },]
         }],
      "protocol": "HTTP_1_1",
      "dataFormat": "JSON",
      "securityMethods": [
        "OAUTH",
      ],
      "interfaceDescriptions": [
        {
          "ipv4Addr":”X.X.X.X",
          "port": NNNN,
          "securityMethods": [
            "OAUTH"
          ]
        }
      ]
    }
  ],
  "description": "MEC App Registration API",
  "supportedFeatures": "1ff",
  "shareableInfo": {
    "isShareable": true,
    "capifProvDoms": [
      "MEC Platform"
    ]
  },
  "serviceAPICategory": "MEC Service Management",
  "apiProvName": "OCF_MEC"
}
```

Upon successful API publication, CAPIF returns the location of the created resource in the Location field of the header and provides the service API ID of the published API in the response body.

This contribution provides a structured approach for onboarding and publishing APIs for the MEC platform within CAPIF, ensuring alignment with CAPIF's standard onboarding and security procedures.

6.7 Open-source implementation of CAPIF

As stated in Annex B.2.2.3 of 3GPP TS 23.222 [2], the NEF can implement the functionalities of the API provider domain functions, as reproduced in Figure 6.7-1.

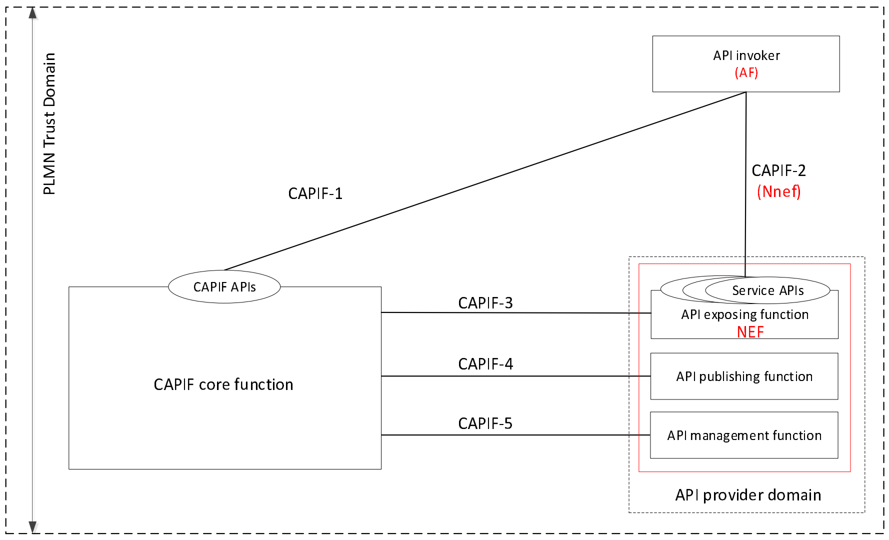

Figure 6.7-1: NEF implements the service specific aspect compliant with the CAPIF architecture

Evolved-5G [7] (ICT-41-2020 project from call H2020-ICT-2018-2020) developed an open-source project of the CAPIF Release 17 under the Apache-2.0 license. EVOLVED-5G implementation of CAPIF works with an open-source implementation of a NEF emulator available in [8]. The NEF emulator used for Evolved-5G implementation supports two services, namely Nnef_EventExposure and Nnef_AFsessionWithQoS [8].

Evolved-5G implementation has been used as the seed code for OpenCAPIF SDG [9] created by ETSI in January 2024.

OpenCAPIF has adopted the following APIs from CAPIF:

- CAPIF API Invoker Management API
- CAPIF API Provider Management API
- CAPIF Publish Service API
- CAPIF Discover Service API
- CAPIF Security API, including obtaining the OAuth 2.0 authorization information with JWT access tokens
- CAPIF Access Control Policy API
- CAPIF Logging API Invocation API
- CAPIF Events API
- CAPIF Auditing API

The implementation also includes a Testing Suite of the above services.

Every CAPIF API is implemented in a separate container for future scalability adaptation. All APIs are exposed and support mutual TLS authentication as specified in 3GPP TS 33.122 [4].

The Source Code of OpenCAPIF and the instructions to deploy it are included in the ETSI GitLab repository [10].

Annex C provides more details of Open-source implementation of CAPIF for NEF publishing API and API consumption from an API Invoker.

7 Summary

This guide presents a number of use cases and adoption strategies for CAPIF. Several stakeholders (e.g., developers, aggregators, CSPs, etc.) can compare them with their own use cases to understand how to apply/map CAPIF capabilities to their own needs.

Annex A:
Examples from OpenCAPIF

All the requests shown in this document for OpenCAPIF can be tested using the OpenCAPIF Postman collection [12]. The collection includes a guide on its use and how to correctly test each of the requests. The following table shows the mapping between each of the requests presented in this document to the corresponding request in the OpenCAPIF Postman collection.

Table A-1: The OpenCAPIF Postman collection

Request	**Section**	**OCF Postman Example**
CAPIF_API_Invoker_Management_API	6.8.1	https://ocf.etsi.org/documentation/develop/testing/postman/#07-onboard_invoker
CAPIF_API_Provider_Management_API	6.4.1	https://ocf.etsi.org/documentation/develop/testing/postman/#04-onboard_provider
CAPIF_Publish_Service_API	6.4.3	https://ocf.etsi.org/documentation/develop/testing/postman/#05-publish_api
CAPIF_Discover_Service_API	6.8.3	https://ocf.etsi.org/documentation/develop/testing/postman/#08-discover
CAPIF_Security_API create context	6.8.5	https://ocf.etsi.org/documentation/develop/testing/postman/#09-security_context
CAPIF_Security_API authorization info	6.8.7	https://ocf.etsi.org/documentation/develop/testing/postman/#10-get_token
API Invocation	6.8.9	https://ocf.etsi.org/documentation/develop/testing/postman/#11-call_service

Annex B: CAPIF Vendor Extensibility

B.1 Include Vendor Specific information

An API provider can include a vendor specific attribute in the ServiceAPIDescription model by enabling "VendorExt" supported feature option and adding the field in the following manner (see clause 5.2.13.2 of 3GPP TS 29.122 [14]):

```
"vendorSpecific-<vendor-identification>": {

      ...

}
```

where "vendor-identification" represents a unique name of the vendor publishing the API, using one of the following structures:

a) IANA-assigned enterprise code (e.g., `vendorSpecific-010415` for "3GPP")
b) Domain name (e.g., `vendorSpecific-3gpp.org`)
c) URN (e.g., `vendorSpecific-urn:3gpp:example`)

B.2 Query for Vendor Specific information

An API invoker can discover API services including vendor specific attributes by adding the following query parameters (see clause 5.2.13.3 of 3GPP TS 29.122 [14]):

a) `supported-features`, enabling "`VendSpecQueryParams`" option
b)
```
vend-spec-<query-parameter-name> = {
      "target": "...",
      "value": "..."

}
```

where "`query-parameter-name`" represents the actual name of the query parameter, "`target`" is a JSON pointer towards the targeted attribute in the targeted resource representation and "`value`" is the actual value of the query parameter to be used for filtering.

B.3 ETSI MEC extensibility to CAPIF

According to ETSI GS MEC 011 V3.3.1 [5] specification, MEC framework, leveraging CAPIF extensibility feature, can engage CAPIF and re-use its functionalities. A concrete example is presented in Table B.3-1 and Table B.3-4, describing the request that a MEC platform has to send towards CAPIF to publish an API, as well as a request of an invoker searching an API with MEC specific attributes.

ETSI MEC supports two extensibility attributes on CAPIF:

a) "`vendorSpecific-urn:etsi:mec:capifext:service-info`" in ServiceAPIDescription and ServiceAPIDescriptionPatch data types, and
b) "`vendorSpecific-urn:etsi:mec:capifext:transport-info`" in AefProfile data type

Both attributes are included on the following API Publish request.

Table B.3-1: Publish MEC API with vendor specific attributes in CAPIF (schema)

```
HTTP POST {apiRoot}/published-apis/v1/{apfId}/service-apis

Request body:
{
  "apiName": "string",
  "aefProfiles": [
    {
      "aefId": "string",
      "versions": [
        {
          "apiVersion": "string",
          "expiry": "string",
          "resources": [
            {
              "resourceName": "string",
              "commType": "string",
              "uri": "string",
              "custOpName": "string",
              "operations": ["string"],
              "description": "string"
            }
          ],
          "custOperations": [
            {
              "commType": "string",
              "custOpName": "string",
              "operations": [
                "string"
              ],
              "description": "string"
            }
          ]
        }
      ],
      "interfaceDescriptions": [
        {
          "ipv4Addr": "string",
          "port": "int",
          "securityMethods": [
            "string"
          ]
        }
       ],
       "vendorSpecific-urn:etsi:mec:capifext:transport-info": {
            "name": "string",
            "description": "string",
            "type": "string",
            "protocol": "string",
            "version": "string",
            "security": {
                "grantTypes": "string",
                "tokenEndpoint": "string"
            }
        }
    }
  ],
  "vendorSpecific-urn:etsi:mec:capifext:service-info": {
       "serializer": "string",
       "state": "string",
       "scopeOfLocality": "string",
       "consumedLocalOnly": "boolean",
       "isLocal": "boolean",
       "category": {
            "href": "string",
            "id": "string",
            "name": "string",
            "version": "string"
       }
  },
  "description": "string",
  "supportedFeatures": "string",
  "apiProvName": "string"
}
```

When APIs are published including Vendor Specific information, API Invokers can filter the Discovery of APIs including query parameters to filter by vendor specific information. Here we show some examples of CAPIF Discover API including vendor specific filtering.

The schema for discovering APIs using vendor specific information is shown in Table B.3-2 and Table B.3-3:

Table B.3-2: Discover APIs with vendor specific query parameters (schema for transport-info)

```
HTTP GET {apiRoot}/service-apis/v1/allServiceAPIs?
    api-invoker-id="string"&
    supported-features="string"&
    vend-spec-type={"target": "...", "value": "string"}
```

Table B.3-3: Discover APIs with vendor specific query parameters (schema for service-info)

```
HTTP GET {apiRoot}/service-apis/v1/allServiceAPIs?
    api-invoker-id="string"&
    supported-features="string"&
    vend-spec-state={"target": "...", "value": "string"}
```

Based in these schemas, the following examples show how to use CAPIF Extensibility. Table B.3-4 shows an API Publication that includes both `transport-info` and `service-info` as part of the API description published.

Table B.3-4: Publish MEC API with vendor specific attributes in CAPIF (OCF example)

```
HTTP POST https://capif.mobilesandbox.cloud:37211/published-apis/v1/{apfId}/service-
apis
```

Request body:

```
{
    "apiName": "api_demo_v2",
    "aefProfiles": [
      {
        "aefId": "AEFb1a522967511fd6d6def0cbb3b5f05",
        "versions": [
          {
            "apiVersion": "v2",
            "expiry": "2021-11-30T10:32:02.004Z",
            "resources": [
              {
                "resourceName": "endpoint_1",
                "commType": "REQUEST_RESPONSE",
                "uri": "/endpoint",
                "custOpName": "string",
                "operations": [
                  "GET"
                ],
                "description": "Endpoint to receive a welcome message"
              }
            ],
            "custOperations": [
              {
                "commType": "REQUEST_RESPONSE",
                "custOpName": "string",
                "operations": [
                  "GET"
                ],
                "description": "string"
              }
            ]
          }
        ],
        "interfaceDescriptions": [
          {
            "ipv4Addr": "localhost",
            "port": 8089,
            "securityMethods": ["Oauth"]
          }
        ],
        "vendorSpecific-urn:etsi:mec:capifext:transport-info": {
            "name": "trasport1",
            "description": "Transport Info 1",
            "type": "REST_HTTP",
            "protocol": "HTTP",
            "version": "2",
            "security": {
                "grantTypes": "OAUTH2_CLIENT_CREDENTIALS",
                "tokenEndpoint": "https://token-endpoint.example.com/"
            }

        }
      }
    ],
    "vendorSpecific-urn:etsi:mec:capifext:service-info": {
        "serializer": "JSON",
        "state": "ACTIVE",
        "scopeOfLocality": "MEC_SYSTEM",
        "consumedLocalOnly": "True",
        "isLocal": "True",
        "category": {
            "href": "https://location.example.com",
            "id": "Location",
            "name": "Location",
            "version": "1.0"
        }
    },
    "description": "Hello api services",
    "supportedFeatures": "4",
    "apiProvName": "OCF_MEC"
  }
```

The next example in Table B.3-5 shows how to query CAPIF using `transport-info` to query for APIs which `vend-spec-type` is “REST_HTTP”.

Table B.3-5: Discover APIs with vendor specific query parameters (OCF example for transport-info)

```
HTTP GET https://capif.mobilesandbox.cloud:37211/service-apis/v1/allServiceAPIs?
    api-invoker-id=INVd4d70e16c52bace2b8f40ed6f6badS&
    supported-features=4&
    vend-spec-type={"target": "/vendorSpecific-urn:etsi:mec:capifext:transport-info",
"value": "REST_HTTP"}
```

Similarly, the next example in Table B.3-6 shows how to query CAPIF for APIs that contains `service-info` which `vend-spec-state` is “ACTIVE”.

Table B.3-6: Discover APIs with vendor specific query parameters (OCF example for service-info)

```
HTTP GET https://capif.mobilesandbox.cloud:37211/service-apis/v1/allServiceAPIs?
    api-invoker-id=INVd4d70e16c52bace2b8f40ed6f6badS&
    supported-features=4&
    vend-spec-state={"target": "/vendorSpecific-urn:etsi:mec:capifext:service-info",
"value": "ACTIVE"}
```

Annex C: Open-source implementation of CAPIF

C.1 NEF Publishes an API

The 5GS network exposure function (NEF) can use CAPIF capabilities to publish and manage its APIs, including managing charging. One of several possible deployment arrangements is shown in Figure C.1-1.

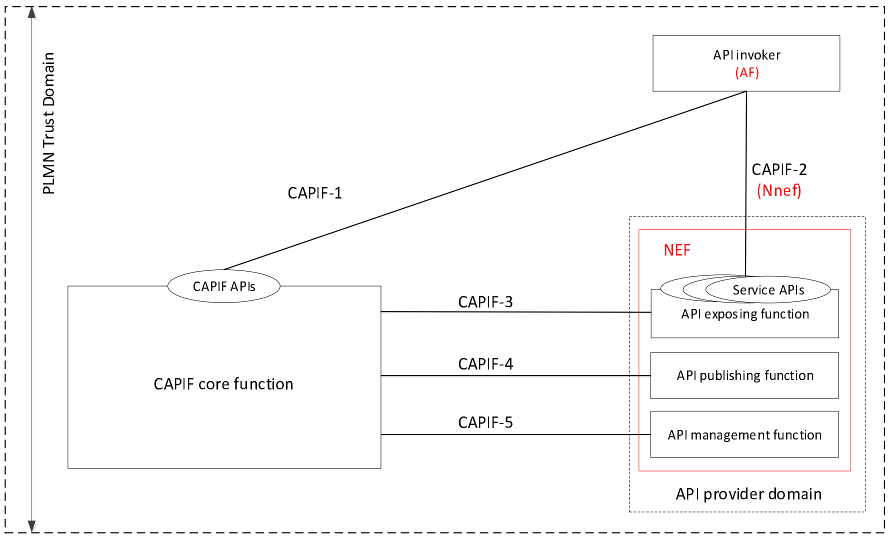

Figure C.1-1: NEF implements the CAPIF API provider domain functions

Signalling for the NEF to publish its APIs to the CAPIF core function is shown in Figure C.1-2, followed by tables detailing the mandatory attributes of the associated request and response schema and example content for those requests and responses, which is based on publicly available examples provided by the OpenCAPIF [9].

A pre-condition is that the NEF (API management function) has been made aware of the appropriate CAPIF core function endpoint (`apiRoot`) with which to initiate the registration, as well as having been provided with a token to access the CAPIF core function.

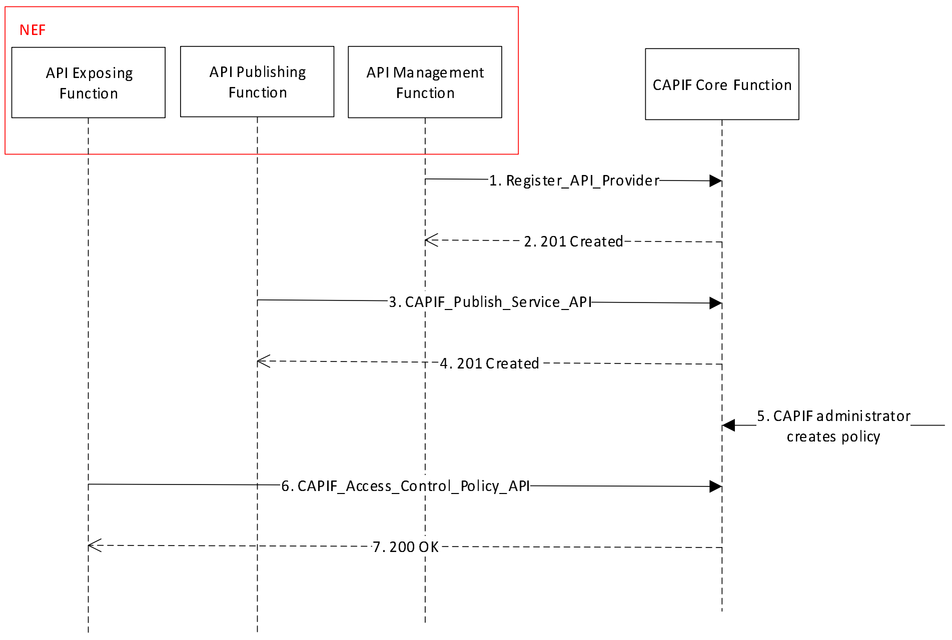

Figure C.1-2: Publish a service API

Details of the signalling flow in Figure C.1-2 are as follows:

NOTE 1: The JSON schema and example snippets provided in the following clauses are aligned with 3GPP TS 29.222 [3], in which `apiVersion` is specified as "`v1`". The example snippets themselves can be tested using the OpenCAPIF Postman collection presented in Annex A and also in the OpenCAPIF Postman documentation [12].

1. CAPIF_API_Provider_Managment_API

The API provider includes:

- Authorization HTTP header (access token)
- Security information enabling the CCF to validate the registration (`regSec`)
- An array (`apiProvFuncs`) of at least one API provider domain function profiles
- Optionally API provider domain information (`apiProvDomInfo`) and the API provider name (`apiProvName`)

Table C.1-1: Register API_provider request (schema)

```
HTTP POST {apiRoot}/api-provider-management/v1/registrations

Request headers:
Authorization: "string"

Request body:
{
  "regSec": "string",
  "apiProvFuncs": [
    {
      "regInfo": {
        "apiProvPubKey": "string",
      },
      "apiProvFuncRole": "string",
      "apiProvFuncInfo": "string"
    }
  ],
  "apiProvDomInfo": "string",
  "suppFeat": "string",
  "apiProvName": "string"
}
```

Given the schema provided in Table C.1-1, an example of the API Provider registration process using OpenCAPIF open-source implementation is now described and then provided in Table C.1-1-OCF. The presented traces contain actual values to illustrate the usage of the information elements in the JSON schema.

NOTE 2: The security related information provided in the examples are truncated for readability, e.g., the value associated with `regSec` and `apiProvPubKey`.

Before carrying out onboarding, the provider requires the:

- OpenCAPIF access token, that will allow the creation of the provider.
- CA_ROOT certificate, necessary to validate the Certificates in all requests.
- URL to which to make the request, i.e., <{apiRoot}/api-provider-management/v1/registrations>.

NOTE 3: In OpenCAPIF, all this information is obtained in a dedicated user registration process [13].

In the presented example, the OpenCAPIF instance is deployed at capif.mobilesandbox.cloud:37211. This instance URL is assumed throughout the presented example traces.

To carry out the onboarding of a provider, the API Provider is required to provide the following fields:

- `regSec`: Contains the OpenCAPIF access token obtained during registration process for the creation of the provider. The Token is also provided in the Authorization HTTP Header.
- `apiProvFuncRole`: Type of provider function to be created. It can be AMF, APF or AEF.
- `apiProvPubKey`: It is the certificate signing request (CSR) of the Provider's function that OpenCAPIF will use to issue the provider's valid certificates. Each function needs to include its own CSR to request a valid Certificate.
- `apiProvFuncInfo`: Information about the provider function that is being created.
- `apiProvDomInfo`: Information about the provider domain that is being created.

The API Provider onboarding request example is shown in Table C.1-1-OCF. The example request is to onboards an API Provider with three functions: AEF, APF and AMF. Therefore, 3 CSRs are provided to obtain 3 Certificates.

Table C.1-1-OCF: Register API provider request (OCF example)

```
HTTP POST https://capif.mobilesandbox.cloud:37211/api-provider-
management/v1/registrations

Request headers:
Authorization: Bearer eyJhbGci0iJSUzIlNiIsInR5cCI6IkpXVCJ9
Content-Type: application/json

Request body:
{
  "regSec": "eyJhbGci0iJSUzIlNiIsInR5cCI6IkpXVCJ9",
  "apiProvFuncs": [
    {
      "regInfo": {
        "apiProvPubKey": "-----BEGIN CERTIFICATE REQUEST-----
\nMIICeTCCAWECAQAWNDEMMAoGA1UEAwwDQUGMRcwFQYDVQQ\n-----END CERTIFICATE REQUEST-----\n"
      },
      "apiProvFuncRole": "AEF",
      "apiProvFuncInfo": "AEF for NEF example"
    },
    {
      "regInfo": {
        "apiProvPubKey": "-----BEGIN CERTIFICATE REQUEST-----
\nMIICeTCCAWECAQAWNDEMMAoGA1UEAwwDQUGMRcwFQYDVQQ\n-----END CERTIFICATE REQUEST-----\n"
      },
      "apiProvFuncRole": "APF",
      "apiProvFuncInfo": "APF for NEF example"
    },
    {
      "regInfo": {
        "apiProvPubKey": "-----BEGIN CERTIFICATE REQUEST-----
\nMIICeTCCAWECAQAWNDEMMAoGA1UEAwwDQUGMRcwFQYDVQQ\n-----END CERTIFICATE REQUEST-----\n"
      },
      "apiProvFuncRole": "AMF",
      "apiProvFuncInfo": "AMF for NEF example"
    }
  ],
  "apiProvDomInfo": "NEF example",
  "apiProvName": "OCF_NEF"
  "suppFeat": "3",
}
```

Upon getting this Request, Open CAPIF core function assigns identity IDs for the API provider (`apiProvDomId`) and each of its functions (`apiProvFuncId`) and returns them in the response (see next section). Furthermore, it provides certificates (`apiProvCert`) for each of the API provider functions based on each function's CSR (`apiProvPubKey`).

2. 201 Created

The API provider domain has been registered successfully. The URI of the created resource is returned in the "Location" HTTP header, including the assigned registration identifier (`registrationId`).

Table C.1-2: Register API provider response (schema)

```
201
Response headers:
Location: {apiRoot}/api-provider-management/v1/registrations/{registrationId}
Response body:
{
  "apiProvDomId": "string",
  "regSec": "string",
  "apiProvFuncs": [
    {
      "apiProvFuncId": "string",
      "regInfo": {
        "apiProvPubKey": "string",
        "apiProvCert": "string"
      },
      "apiProvFuncRole": "string",
      "apiProvFuncInfo": "string"
    }
  ],
  "apiProvDomInfo": "string",
  "suppFeat": "string",
  "apiProvName": "string"
}
```

The response received from the successful onboarding request using OpenCAPIF project for the creation of a new API Provider includes the following fields:

- `apiProvDomId`: Identifier of the created provider, which is also the ID returned as part of the 'Location' header.

- `apiProvCert:` Certificate of the provider function created from the CSR sent in the onboarding request, which will be used for subsequent requests from the provider towards OpenCAPIF. Each function is provided with its own Certificate (i.e., one each for the AEF, APF and AMF functions).

- `apiProvFuncId:` Identifier of the provider function created.

The example Register API provider response is shown in Table C.1-2-OCF.

Table C.1-2-OCF: Register API provider response (OCF example)

```
Response headers:
Location: https://capif.mobilesandbox.cloud:37211/api-provider-
management/v1/registrations/3a4df5f6e30a5ee34de346004ee6bb

Response body:
{
  "apiProvDomId": "3a4df5f6e30a5ee34de346004ee6bb",
  "regSec": "eyJhbGci0iJSUzIlNiIsInR5cCI6IkpXVCJ9",
  "apiProvFuncs": [
    {
      "apiProvFuncId": "AEFb1a522967511fd6d6def0cbb3b5f05",
      "regInfo": {
        "apiProvPubKey": "-----BEGIN CERTIFICATE REQUEST-----
\nMIICeTCCAWECAQAWNDEMMAoGA1UEAwwDQUGMRcwFQYDVQQ\n-----END CERTIFICATE REQUEST-----\n",
        "apiProvCert": "-----BEGIN CERTIFICATE -----
\nMIIDRTCCA12gAwIBAgIUC2D11wSISz1CjKiFXJ3vwKxVhREwDQYJKoZ\n-----END CERTIFICATE-----\n"
      },
      "apiProvFuncRole": "AEF",
      "apiProvFuncInfo": "AEF for NEF example"
    },
    {
      "apiProvFuncId": "APF54f49ee44f67c81fc7d7f321195531",
      "regInfo": {
        "apiProvPubKey": "-----BEGIN CERTIFICATE REQUEST-----
\nMIICeTCCAWECAQAWNDEMMAoGA1UEAwwDQUGMRcwFQYDVQQ\n-----END CERTIFICATE REQUEST-----\n",
        "apiProvCert": "-----BEGIN CERTIFICATE -----
\nMIIDRTCCA12gAwIBAgIUC2D11wSISz1CjKiFXJ3vwKxVhREwDQYJKoZ\n-----END CERTIFICATE-----\n"
      },
      "apiProvFuncRole": "APF",
      "apiProvFuncInfo": "APF for NEF example"
    },
    {
      "apiProvFuncId": "AMFOdbe115f6759f0fec943321dc2184d",
      "regInfo": {
        "apiProvPubKey": "-----BEGIN CERTIFICATE REQUEST-----
\nMIICeTCCAWECAQAWNDEMMAoGA1UEAwwDQUGMRcwFQYDVQQ\n-----END CERTIFICATE REQUEST-----\n",
        "apiProvCert": "-----BEGIN CERTIFICATE -----
\nMIIDRTCCA12gAwIBAgIUC2D11wSISz1CjKiFXJ3vwKxVhREwDQYJKoZ\n-----END CERTIFICATE-----\n"
      },
      "apiProvFuncRole": "AMF",
      "apiProvFuncInfo": "AMF for NEF example"
    }
  ],
  "apiProvDomInfo": "NEF example",
  "apiProvName": "OCF_NEF"
  "suppFeat": "3",
}
```

3. CAPIF_Publish_Service_API

The API publishing function sends information on the offered APIs by providing the API exposing function profiles array (`aefProfiles`). The published APIs resource created in response to the request represents all published service APIs of the NEF.

NOTE 4: The example content in Table C.1.3, the attribute `apiStatus` is omitted, implying the Service API is active at all AEF(s) present in the `aefProfiles` attribute. Furthermore, it is assumed the exchange is over CAPIF-4, rather than CAPIF-6/6e. As a consequence, the `pubApiPath` and `ccfId` attributes are omitted.

Table C.1-3: Publish API request (schema)

```
HTTP POST {apiRoot}/published-apis/v1/{apfId}/service-apis
Request body:
{
  "apiName": "string",
  "apiStatus": {
    "aefIds": [
      "string"
    ]
  },
  "aefProfiles": [
    {
      "aefId": "string",
      "versions": [
        {
          "apiVersion": "string",
          "expiry": "string",
          "resources": [
            {
              "resourceName": "string",
              "commType": "string",
              "uri": "string",
              "custOpName": "string",
              "operations": ["string"],
              "description": "string"
            },
            {
              "resourceName": "string",
              "commType": "string",
              "uri": "string",
              "custOpName": "string",
              "operations": ["string"],
              "description": "string"
            }
          ],
          "custOperations": [
            {
              "commType": "string",
              "custOpName": "string",
              "operations": [
                "string"
              ],
              "description": "string"
            }
          ]
        }
      ],
      "protocol": "string",
      "dataFormat": "string",
      "securityMethods": [
        "string",
      ],
      "interfaceDescriptions": [
        {
          "ipv4Addr": "string",
          "port": "int",
          "securityMethods": [
            "string"
          ]
        }
      ]
    }
  ],
  "description": "string",
  "supportedFeatures": "string",
  "shareableInfo": {
    "isShareable": "boolean",
    "capifProvDoms": [
      "string"
    ]
  },
  "serviceAPICategory": "string",
  "apiSuppFeats": "string",
  "pubApiPath": {
    "ccfIds": [
      "string"
    ]
  },
  "ccfId": "string",
  "apiProvName": "string"
}
```

To publish a service using OpenCAPIF, it is necessary to specify the following fields:

- `apiName:` Name of the API to be published, which must be unique in CAPIF.
- `apiStatus`: Represents the API status. In addition, if this attribute is omitted, it is understood that the API is available in all AEFs of `aefProfiles`. It contains the `aefIds` field which is a list of the AEFs that expose the API. If this list is empty or the attribute is omitted, it is understood that the API is inactive in all `aefProfiles.`
- `aefProfiles:` List of Information about AEFs that expose the API. This information contains the following fields:
    - `aefId:` AEF identifier of the provider that exposes the service.
    - `versions`: List of information about API versions. Each version contains the resources supported by the API. Each version contains:
        - `apiVersion`: usable version of the API.
        - `expiry:` date on which the API version expires and will no longer be available.
        - `resources`: A list of resources that contain the API version. Each resource contains the following information:
            - `resourceName`: Name of the resource for which its information is specified.
            - `commType`: Type of use of the resource, which can be receiving a response when making a request (REQUEST_RESPONSE) or a subscription to receive notifications (SUBSCRIBE_NOTIFY)
            - `uri`: URI to make the request to use the described API resource.
            - `custOpName`: Part of the URI structure used for a custom operation associated with the resource. The attributes `custOpName` and `custOperations` are mutually exclusive.
            - `custOperations`: List of the resource's custom operations associated to this resource. The attributes `custOpName` and `custOperations` are mutually exclusive.
            - `operations`: List of HTTP methods supported by the API resource.
            - `description`: Descriptive text of the API resource.
        - `custOperations`: List of custom operations without resource association.
    - `protocol:` Indicates a protocol and protocol version used by the API.
    - `dataFormat:` Indicates the data sending format which can be JSON, XML or PROTOBUF3.
    - `securityMethods:` Indicates the possible security methods of the API.
    - `domainName:` Domain name to which the API belongs.
    - `interfaceDescriptions:` List of interfaces to which API requests can be made.
      Only `domainName` or `interfaceDescription` can be specified.

- `supportedFeatures:` Indicates the one or more CAPIF API specific supported features (from the nine features specified in 3GPP TS 29.222 [3]), e.g., support for API status monitoring in the CAPIF layer as part of the SEAL framework, support for RNAA functionality.

- `shareableinfo:` Indicates whether the service API and/or the service API category can be shared to the list of CAPIF provider domains.

- `serviceAPICategory:` The service API category to which the service API belongs to.

- `apiSuppFeats:` Indication provided by the consumer (i.e., related to the AEF) which optional features are supported by the service API.

- `apiProvName:` API provider Name that publishes the API.

The example API Publish request to publish “3gpp-as-session-with-qos” API from NEF is shown in Table C.1-3-OCF, for which there are twenty-nine optional service specific features (`apiSuppFeats`) specified in 3GPP TS 29.122 [14].

Table C.1-3-OCF: Publish API request (OCF example)

```
HTTP POST https://capif.mobilesandbox.cloud:37211/published-
apis/v1/APF54f49ee44f67c81fc7d7f321195531/service-apis

Request body:
{
  "apiName": "3gpp-as-session-with-qos",
  "apiStatus": {
    "aefIds": ["AEFb1a522967511fd6d6def0cbb3b5f05"
    ]
  },
  "aefProfiles": [
    {
      "aefId": "AEFb1a522967511fd6d6def0cbb3b5f05",
      "versions": [
        {
          "apiVersion": "v1",
          "expiry": "2100-11-30T10:32:02.004Z",
          "resources": [
            {
              "resourceName": "QOS_SUBSCRIPTIONS",
              "commType": "SUBSCRIBE_NOTIFY",
              "uri": "/{scsAsId}/subscriptions",
              "operations": [
                "GET",
                "POST"
              ],
              "description": "Endpoint to manage monitoring subscriptions"
            },
            {
              "resourceName": "QOS_SUBSCRIPTION_SINGLE",
              "commType": "SUBSCRIBE_NOTIFY",
              "uri": "/{scsAsId}/subscriptions/{subscriptionId}",
              "operations": [
                "GET",
                "PUT",
                "DELETE"
              ],
              "description": "Endpoint to manage single subscription"
            }
          ],
        }
      ],
      "protocol": "HTTP_1_1",
      "dataFormat": "JSON",
      "securityMethods": [
        "OAUTH",
        "PSK"
      ],
      "interfaceDescriptions": [
        {
          "ipv4Addr": "127.0.0.1",
          "port": 4443,
          "securityMethods": [
            "OAUTH"
          ]
        }
      ]
    }
  ],
  "description": "3gpp-as-session-with-qos NEF API",
  "supportedFeatures": "1ff",
  "shareableInfo": {
    "isShareable": true,
    "capifProvDoms": [
      "mobilesandbox.cloud"
    ]
  },
  "serviceAPICategory": "NEF",
  "apiSuppFeats": "1fffffff",
  "apiProvName": "OCF_NEF"
}
```

4. 201 Created

The service API has been published successfully The URI of the created resource is returned in the "Location" HTTP header, including the assigned service API identifier (`serviceApiId`) that is also equal to the API identifier (`apiId`) in the response body.

Table C.1-4: Publish API response (schema)

```
{
201
Response headers:
Location: {apiRoot}/published-apis/v1/{apfId}/service-apis/{serviceApiId}

Response body:
{
  "apiName": "string",
  "apiId": "string",
  "apiStatus": {
    "aefIds": [
      "string"
    ]
  },
  "aefProfiles": [
    "object"
  ],
  "description": "string",
  "supportedFeatures": "string",
  "shareableInfo": {
    "isShareable": "boolean",
    "capifProvDoms": [
      "string"
    ]
  },
  "serviceAPICategory": "string",
  "apiSuppFeats": "string",
  "pubApiPath": {
    "ccfIds": [
      "string"
    ]
  },
  "ccfId": "string"
  "apiProvName": "string"
}
```

The response to a successful API Publish request using OpenCAPIF contains the following information:

- `Location`: URL with the resource created for the API Published.
- `apiId`: The assigned Identifier to the Published API. It is required for the UPDATE or DELETE API procedures once the API has been published.

The information from the API that was sent in the request will be obtained as a response with a new field, the "`apiId`", which is the identifier of the API that has been created and will enable subsequent modify or delete procedures.

Table C.1-4-OCF: Publish API response (OCF example)

Response headers:
```
Location: https://capif.mobilesandbox.cloud:37211/published-
apis/v1/APF54f49ee44f67c81fc7d7f321195531/service-apis/f55731ec5f8ce703ac1f69605ad095
```

Response body:
```
{
  "apiName": "3gpp-as-session-with-qos",
  "apiId": "f55731ec5f8ce703ac1f69605ad095",
  "aefProfiles": [
    {
      "aefId": "AEFb1a522967511fd6d6def0cbb3b5f05",
      "versions": [
        {
          "apiVersion": "v1",
          "expiry": "2100-11-30T10:32:02.004Z",
          "resources": [
            {
              "resourceName": "QOS_SUBSCRIPTIONS",
              "commType": "SUBSCRIBE_NOTIFY",
              "uri": "/{scsAsId}/subscriptions",
              "custOpName": "http_post",
              "operations": [
                "GET",
                "POST"
              ],
              "description": "Endpoint to manage monitoring subscriptions"
            },
            {
              "resourceName": "QOS_SUBSCRIPTION_SINGLE",
              "commType": "SUBSCRIBE_NOTIFY",
              "uri": "/{scsAsId}/subscriptions/{subscriptionId}",
              "operations": [
                "GET",
                "PUT",
                "DELETE"
              ],
              "description": "Endpoint to manage single subscription"
            }
          ],
        }
      ],
      "protocol": "HTTP_1_1",
      "dataFormat": "JSON",
      "securityMethods": [
        "OAUTH",
        "PSK"
      ],
      "interfaceDescriptions": [
        {
          "ipv4Addr": "127.0.0.1",
          "port": 4443,
          "securityMethods": [
            "OAUTH"
          ]
        }
      ]
    }
  ],
  "description": "3gpp-as-session-with-qos NEF API",
  "supportedFeatures": "1ff",
  "shareableInfo": {
    "isShareable": true,
    "capifProvDoms": [
      "mobilesandbox.cloud"
    ]
  },
  "serviceAPICategory": "NEF",
  "apiSuppFeats": "1fffffff"
  "apiProvName": "OCF_NEF"
}
```

5. CAPIF administrator creates policy

The procedures for creation of the access control policy at the CAPIF core function by the CAPIF administrator is outside the scope of 3GPP TS 23.222 [2] and can be triggered because of the API management function issuing the "Register API provider" request or prior to that. Either way, sufficient information must be provided to enable the AEF specific access control policy list resource for the service API to be created. Specifically, each policy must include the CCF assigned API invoker identifier (`apiInvokerId`) and can include allowed total invocations (`allowedTotalInvocations`), invocations per second (`allowedInvocationsPerSecond`), invocation time range list (`allowedInvocationTimeRangeList`). Since an API invoker identifier is mandatory, it is a prerequisite that API invoker has been onboarded prior to invoking the highlighted access control policy procedures.

6. CAPIF_Access_Control_Policy_API

The Access Control List (ACL) service allows the provider to use AEF to obtain the access control list from invokers for an exposing API. The provider is in charge of making the request to the CAPIF Core Function and using the list received to control an invoker's access to one of its APIs.

Table C.1-6: GET access control policy request (schema)

```
HTTP GET {apiRoot}/access-control-policy/v1/accessControlPolicyList/{serviceApiId}?aef-
id="string"
```

The AEF may make a request to OpenCAPIF to obtain all the ACLs of a service that it is exposing, for this it must specify in the URL of the request the identifier of the service (`apiId` received in the API Publish response) for which it wants the list of ACLs, in addition to the AEF’s own identifier to check that it has access to that list.

Table C.1-6-OCF: ACL Request (OCF example)

```
HTTP GET https://capif.mobilesandbox.cloud:37211/access-control-
policy/v1/accessControlPolicyList/f55731ec5f8ce703ac1f69605ad095?aef-
id=AEFb1a522967511fd6d6def0cbb3b5f05
```

7. 200 OK

The request has been successfully received and the CAPIF Core Function returns the invoker access control information for the requested API. The AEF will use this information to allow or not the use of the API by the invoker.

Table C.1-7: GET access control policy response (schema)

```
200

Response body:
{
  "apiInvokerPolicies": [
    {
      "apiInvokerId": "string",
      "allowedTotalInvocations": "integer",
      "allowedInvocationsPerSecond": "integer",
      "allowedInvocationTimeRangeList": [
        {
          "startTime": "string",
          "stopTime": "string"
        }
      ]
    }
  ]
}
```

OpenCAPIF provides the access control policy information for the NEF API Published "3gpp-as-session-with-qos" containing the following information:

- `apiInvokerId:` identifier of the invoker to which the access policy belongs.
- `allowedTotalInvocations:` Total number of invocations allowed on the service API by the API invoker.
- `allowedInvocationsPerSecond:` invocations per second allowed on the service API by the API invoker.
- `allowedInvocationTimeRangeList:` the time ranges during which the invocations are allowed on the service API by the API invoker.

Table C.1-7-OCF: GET access control policy response (OCF example)

```
200

Response body:
{
  "apiInvokerPolicies": [
    {
      "apiInvokerId": "INVd4d70e16c52bace2b8f40ed6f6badS",
      "allowedTotalInvocations": 1200,
      "allowedInvocationsPerSecond": 2,
      "allowedInvocationTimeRangeList": [
        {
          "startTime": "2024-08-19T09:00:00.000000+02:00",
          "stopTime": "2024-08-23T16:00:00.000000+02:00"
        }
      ]
    }
  ]
}
```

8. Offboarding API Provider

The provider can request its deletion from CAPIF using the offboarding service. By using this service, all information stored in the CAPIF Core Function related to the provider will be deleted, including APIs published by the provider, security contexts of those APIs, ACLs, etc.

Table C.1-8: DELETE API provider request (schema)

```
HTTP DELETE {apiRoot}/api-provider-management/v1/registrations/{registrationId}
```

For the process of deleting a Provider in OpenCAPIF, you will need the AMF certificate and the identifier of the provider that you want to delete. This data is obtained in the provider creation process and by simply making the corresponding request we will eliminate the provider.

Table C.1-8-OCF: DELETE API provider request (OCF example)

```
HTTP DELETE https://capif.mobilesandbox.cloud:37211/api-provider-
management/v1/registrations/3a4df5f6e30a5ee34de346004ee6bb
```

9. 204 No Content

The response will be a 204 No Content, which indicates that the provider has been successfully deleted along with everything related to it, APIs Published, security contexts associated with its services, ACLs of its services, etc.

C.2 API consumption from an API Invoker

This section explains how an API Invoker can use OpenCAPIF [9] to discover and consume APIs, based on 3GPP TS 23.222 [2] and 3GPP TS 29.222 [3] for the schema. The API Invoker application needs to obtain the following information from OpenCAPIF during user registration, similarly to the API Provider onboarding:

- `CA_ROOT:` Root certificate from the OpenCAPIF Certificate Authority to validate Certificates.
- `Token:` Token for the API Invoker onboarding request.
- `Onboarding URL:` URL for requesting the API Invoker onboarding.

This section explains:

- how to onboard an API Invoker in OpenCAPIF, to be able to discover all the APIs published in the CAPIF Core Function.
- request the creation of a security context, which is necessary to consume an API.
- request the required OAuth token to consume the API; and finally, how to make the request directly to the published API.

All the requests shown in this document can be tested using the OpenCAPIF Postman collection found in Annex A and in the OpenCAPIF Postman documentation [12].

1. **CAPIF_API_Invoker_Management (Registration)**

Before carrying out onboarding, the invoker will need the CAPIF access token that will allow the creation of the invoker, the CA_ROOT certificate necessary to validate all the requests, and the URL to which to make the request <{apiRoot}/api-invoker-management/v1/onboardedInvokers>.

In OpenCAPIF, all this information is obtained in the user registration process [13].

To onboard an API Invoker in CAPIF, it is necessary to utilize the token received in the Authentication HTTP header. In addition, a Certificate Signing Request (CSR) will have to be created with two keys so that the CAPIF Core Function can issue a valid certificate that will be used by the invoker to be able to make the subsequent requests. In the following example, the `{apiList}` field has been omitted, as it will discover the APIs that can be used with the discover service.

Table C.2-1: Register API Invoker request (schema)

```
HTTP POST {apiRoot}/api-invoker-management/v1/onboardedInvokers

Request headers:
Authorization: "string"

Request body:
{
    "onboardingInformation": {
        "apiInvokerPublicKey": "string"
    },
    "notificationDestination": "string",
    "apiInvokerInformation": "string",
    "websockNotifConfig": {
        "websocketUri": "string",
        "requestWebsocketURI": "boolean"
    }
}
```

To perform the onboarding of an API Invoker, the following data will have to be specified:

- `apiInvokerPublicKey`: It is the certificate signing request of the invoker that CAPIF will use to issue the invoker's valid certificate.
- `apiInvokerInformation`: Generic information related to the API invoker such as details of the device or the application
- `notificationDestination`: URL where the API invoker can receive CAPIF notifications.

Table C.2-1-OCF: Register API Invoker request (OCF example)

```
HTTP POST https://capif.mobilesandbox.cloud:37211/api-invoker-
management/v1/onboardedInvokers/

Request headers:
Authorization: Bearer eyJhbGci0iJSUzIlNiIsInR5cCI6IkpXVCJ9
Content-Type: application/json

Request body:
{
    "onboardingInformation": {
        "apiInvokerPublicKey": "-----BEGIN CERTIFICATE REQUEST-----
\nMIICeTCCAWECAQAHISNoGA1ERGAwwDQUGMRcwFQYDVQQ\n-----END CERTIFICATE REQUEST----\n"
    },
    "notificationDestination": " http://nef_notification_destination:8080/",
    "apiInvokerInformation": "NEF Invoker",
    "websockNotifConfig": {
        "websocketUri": "websocketUri",
        "requestWebsocketURI": true
    }
}
```

The CAPIF Core Function assigns an identity for the API invoker (`apiInvokerId`) and a certificate (`apiInvokerCertificate`) based on provided public key(`apiInvokerPublicKey`).

2. 201 Created

The API Invoker has been registered successfully. The URI of the created resource is returned in the "Location" HTTP header including the assigned registration identifier (`apiInvokerId`).

Table C.2-2: Register API-Invoker response (schema, then example content)

```
201
Response headers:
Location: {apiRoot}/api-invoker-management/v1/onboardedInvokers/{apiInvokerId}

Response body:
{
    "apiInvokerId": "string",
    "onboardingInformation": {
        "apiInvokerPublicKey": "string",
        "apiInvokerCertificate": "string",
        "onboardingSecret": "string"
    },
    "notificationDestination": "string",
    "apiInvokerInformation": "string",
    "websockNotifConfig": {
        "websocketUri": "string",
        "requestWebsocketURI": "boolean"
    }
}
```

The fields that are returned when onboarding an invoker, that are populated by the CCF, are:

- `apiInvokerId`: Identifier of the invoker created.
- `apiInvokerCertificate`: Certificate of the Invoker created from the CSR sent in the onboarding request, which will be used for the following requests from the Invoker to CAPIF Core Function.

Table C.2-2-OCF: Register API-Invoker response (OCF example)

```
Response headers:
Location: https://capif.mobilesandbox.cloud:37211/api-invoker-
management/v1/onboardedInvokers/INVaa03f9911db37be051b243ce3caad9

Response body:
{
    "apiInvokerId": "INVaa03f9911db37be051b243ce3caad9",
    "onboardingInformation": {
        "apiInvokerPublicKey": "-----BEGIN CERTIFICATE REQUEST-----
\nMIICeTCCAWECAQAHISNoGA1ERGAwwDQUGMRcwFQYDVQQ\n-----END CERTIFICATE REQUEST----\n",
        "apiInvokerCertificate": "-----BEGIN CERTIFICATE-----
\nMIIDgjCCAmqgAwIBAgIUS18nvjZYU5B+ptuAw/
x3gQWYEJVxDha9112mP7lclNg5yd5o1Xwj1Imbht2z33ZJ4jxA=\n-----END CERTIFICATE-----\n"
    },

    "apiInvokerInformation": "NEF Invoker",
    "websockNotifConfig": {
        "websocketUri": "websocketUri",
        "requestWebsocketURI": true
    }
}
```

3. CAPIF_Discovery_Service_API

After onboarding the API Invoker, the API Discovery function is used by invokers to obtain all the APIs published in CAPIF, providing all the necessary information that an API Invoker needs to be able to make a request to each such API.

Table C.2-3: Discover APIs request (schema, then example content)

```
HTTP GET {apiRoot}/service-apis/v1/allServiceAPIs?api-invoker-id={apiInvokerId}
```

The API Invoker must make the request to CAPIF using its identifier `{apiInvokerId}` in order to verify that the invoker is registered in CAPIF.

Table C.2-3-OCF: Discover APIs request (OCF example)

```
HTTP GET https://capif.mobilesandbox.cloud:37211/service-apis/v1/allServiceAPIs?api-
invoker-id=INVaa03f9911db37be051b243ce3caad9
```

4. OK 200

Upon successful request, the API Invoker will receive the 200 OK code and all the information about the APIs published in the CAPIF Core Function containing the AEF profiles exposing the APIs. Using this information, the API Invoker can decide which APIs it wants to use.

Table C.2-4-OCF: Discover APIs response (schema)

```
200

Response body:
{
  "serviceAPIDescription":[    {
      "apiName": "string",
      "apiId": "string",
      "apiStatus": {
        "aefIds": [
          "string"
        ]
      },
      "aefProfiles": [
        {
          "aefId": "string",
          "versions": [
            {
              "apiVersion": "string",
              "expiry": "string",
              "resources": [
                {
                  "resourceName": "string",
                  "commType": "string",
                  "uri": "string",
                  "custOpName": "string",
                  "operations": ["string"],
                  "description": "string"
                },
                {
                  "resourceName": "string",
                  "commType": "string",
                  "uri": "string",
                  "custOpName": "string",
                  "operations": ["string"],
                  "description": "string"
                }
              ],
              "custOperations": [
                {
                  "commType": "string",
                  "custOpName": "string",
                  "operations": [
                    "string"
                  ],
                  "description": "string"
                }
              ]
            }
          ],
          "protocol": "string",
          "dataFormat": "string",
          "securityMethods": [
            "string",
          ],
          "interfaceDescriptions": [
            {
              "ipv4Addr": "string",
              "port": "int",
              "securityMethods": [
                "string"
              ]
            }
          ]
        }
      ],
      "description": "string",
      "supportedFeatures": "string",
      "shareableInfo": {
        "isShareable": "boolean",
        "capifProvDoms": [
          "string"
        ]
      },
      "serviceAPICategory": "string",
      "apiSuppFeats": "string",
      "pubApiPath": {
        "ccfIds": [
          "string"
        ]
      },
      "ccfId": "string"
      "apiProvName": "string"
    }
  ]
}
```

The API Invoker will receive a list with the APIs published in CAPIF, including the following information:

- `apiName:` Name of the API to be published, which must be unique in CAPIF.
- `apiStatus:` Represents the API status, in addition, if this attribute is omitted, it is understood that the API is available in all AEFs of `aefProfiles`. It contains the `aefIds` field which is a list of the AEFs that expose the API. If this list is empty or the attribute is omitted, it is understood that the API is inactive in all `aefProfiles.`
- `aefProfiles:` List of Information about AEFs that expose the API. This information contains the following fields:
    - o `aefId:` AEF identifier of the provider that exposes the service.
    - o `versions`: list of information about API versions. Each version contains the resources supported by the API. Each version contains:
        - ▪ `apiVersion`: usable version of the API.
        - ▪ `expiry:` date on which the API version expires and will no longer be available.
        - ▪ `resources`: A list of resources that contain the API version. Each resource contains the following information:
            - `resourceName`: Name of the resource for which its information is specified.
            - `commType`: Type of use of the resource, which can be receiving a response when making a request (REQUEST_RESPONSE) or a subscription to receive notifications (SUBSCRIBE_NOTIFY)
            - `uri`: URI to make the request to use the described API resource.
            - `custOpName`: Part of the URI structure used for a custom operation associated with the resource. The attributes `custOpName` and `custOperations` are mutually exclusive.
            - `custOperations`: List of the resource's custom operations associated to this resource. The attributes `custOpName` and `custOperations` are mutually exclusive.
            - `operations`: list of HTTP methods supported by the API resource.
            - `description`: descriptive text of the API resource.
        - ▪ `custOperations`: List of custom operations without resource association.
    - o `protocol:` indicates a protocol and protocol version used by the API.
    - o `dataFormat:` Indicates the data sending format which can be JSON, XML or PROTOBUF3.
    - o `securityMethods:` Indicates the possible security methods of the API.
    - o `domainName:` Domain name to which the API belongs.
    - o `interfaceDescriptions:` List of interfaces to which API requests can be made.
      Only `domainName` or `interfaceDescription` can be specified.
- `supportedFeatures:` Indicates the one or more CAPIF API specific supported features (from the nine features specified in 3GPP TS 29.222 [3]), e.g., support for API status monitoring in the CAPIF layer as part of the SEAL framework, support for RNAA functionality.
- `shareableinfo:` Indicates whether the service API and/or the service API category can be shared to the list of CAPIF provider domains.
- `serviceAPICategory:` The service API category to which the service API belongs to.
- `apiSuppFeats:` Indication provided by the consumer (i.e., related to the AEF) which optional features are supported by the service API.

- `apiProvName`: API provider Name that publishes the API.

An example for the Discover APIs response for an NEF that exposes the “3gpp-as-session-with-qos” API is shown in Table C.2-4-OCF, for which there are twenty-nine optional service specific features (`apiSuppFeats`) specified in 3GPP TS 29.122 [14].

Table C.2-4-OCF: Discover APIs response (OCF example)

```
200

Response body:
{
 "serviceAPIDescription":[       {
      "apiName": "3gpp-as-session-with-qos",
      "apiId": "f55731ec5f8ce703ac1f69605ad095",
      "aefProfiles": [
        {
          "aefId": "string",
          "versions": [
            {
              "apiVersion": "v1",
              "expiry": "2100-11-30T10:32:02.004Z",
              "resources": [
                {
                  "resourceName": "QOS_SUBSCRIPTIONS",
                  "commType": "SUBSCRIBE_NOTIFY",
                  "uri": "/{scsAsId}/subscriptions",
                  "operations": [
                    "GET",
                    "POST"
                  ],
                  "description": "Endpoint to manage monitoring subscriptions"
                },
                {
                  "resourceName": "QOS_SUBSCRIPTION_SINGLE",
                  "commType": "SUBSCRIBE_NOTIFY",
                  "uri": "/{scsAsId}/subscriptions/{subscriptionId}",
                  "operations": [
                    "GET",
                    "PUT",
                    "DELETE"
                  ],
                  "description": "Endpoint to manage single subscription"
                }
              ],
            }
          ],
          "protocol": "HTTP_1_1",
          "dataFormat": "JSON",
          "securityMethods": [
            "OAUTH",
            "PSK"
          ],
          "interfaceDescriptions": [
            {
              "ipv4Addr": "127.0.0.1",
              "port": 4443,
              "securityMethods": [
                "OAUTH"
              ]
            }
          ]
        }
      ],
      "description": "3gpp-as-session-with-qos NEF API",
      "supportedFeatures": "1ff",
      "shareableInfo": {
        "isShareable": true,
        "capifProvDoms": [
          "mobilesandbox.cloud"
        ]
      },
      "serviceAPICategory": "NEF",
      "apiSuppFeats": "1fffffff",
      "apiProvName": "OCF_NEF"
    }
  ]
}
```

5. CAPIF_Security_API

This request is responsible for requesting the creation of a security context to be able to use an API. If this step were not carried out, even if the API invoker had the information on an API it wished to use, the token giving access to the API would be missing, which can only be obtained once the security context has been created.

Table C.2-5: Create Security Context request (schema)

```
HTTP PUT {apiRoot}/capif-security/v1/trustedInvokers/{apiInvokerId}


Request body:
{
    "securityInfo": [
        {
            "prefSecurityMethods": [
                "string"
            ],
            "aefId": "string",
            "apiId": "string",
            "authenticationInfo": "string",
            "authorizationInfo": "string"
        }
    ],
    "notificationDestination": "string",
    "supportedFeatures": "string"
}
```

In this request we find the following fields:

- `securityInfo:` List of information that represents the details of the interfaces and security methods of the APIs the API invoker wishes to use.
- `prefSecurityMethods:` List of preferred security methods for using an API. In the examples, OAuth tokens are used.
- `aefId:` Identifier of the API exposing function that expose the API.
- `apiId:` Identifier of the API that that the API invoker wishes to use.
- `notificationDestination:` Address where the notifications should be delivered to.

The following example asks for OAUTH as the preferred security method. When this method is selected by the CAPIF Core Function, the API invoker must make another request to obtain the necessary security access token to consume the API.

Table C.2-5-OCF: Create Security Context request (OCF example)

```
HTTP PUT https://capif.mobilesandbox.cloud:37211/capif-
security/v1/trustedInvokers/INVaa03f9911db37be051b243ce3caad9



Request body:
{
    "securityInfo": [
        {
            "prefSecurityMethods": ["OAUTH"],
            "selSecurityMethod": "OAUTH",
            "aefId": "AEFb1a522967511fd6d6def0cbb3b5f05",
            "apiId": "f55731ec5f8ce703ac1f69605ad095",
            "authenticationInfo": "authenticationinfo",
            "authorizationInfo": "authorizationInfo"
        }
    ],
    "notificationDestination": "http://nef_notification_destination:8080/",
    "supportedFeatures": "f"
}
```

OpenCAPIF selects one of the preferred methods to make requests to the API based in AEF profile. In this example, only one Security Method is requested by the API Invoker, so OpenCAPIF checks that this Security Method is supported by the API Exposer.

6. 201 Created

The security context has been created successfully and the security context created with the selected security method is returned in the response body.

Table C.2-6: Create Security Context response (schema)

```
201

Response headers:
Location: {apiRoot}/capif-security/v1/trustedInvokers/{apiInvokerId}

Response body:
{
    "securityInfo": [
        {
            "prefSecurityMethods": [
                "string"
            ],
            "selSecurityMethod": "string",
            "aefId": "string",
            "apiId": "string",
            "authenticationInfo": "string",
            "authorizationInfo": "string"
        }
    ],
    "notificationDestination": "string",
    "supportedFeatures": "string"
}
```

The `{selSecurityMethod}` is returned, which specifies the security method that will be used so that the API Invoker can consume the API and the URI of the created resource in the "Location" HTTP header that includes the `{apiInvokerId}`.

Table C.2-6-OCF: Create Security Context response (OCF example)

```
201

Response headers:
Location: https://capif.mobilesandbox.cloud:3721/capif-
security/v1/trustedInvokers/INVaa03f9911db37be051b243ce3caad9



Response body:
{
    "securityInfo": [
        {
            "prefSecurityMethods": ["OAUTH"],
            "selSecurityMethod": "OAUTH",
            "aefId": "AEFb1a522967511fd6d6def0cbb3b5f05",
            "apiId": "f55731ec5f8ce703ac1f69605ad095",
            "authenticationInfo": "authenticationinfo",
            "authorizationInfo": "authorizationInfo"
        }
    ],
    "notificationDestination": "http://nef_notification_destination:8080/",
    "supportedFeatures": "f"
}
```

7. CAPIF_Security_API (Obtain Token)

When the security method selected is OAUTH in the security context created by the API Invoker, the API Invoker must make a request to the CAPIF Core Function to obtain the access token to consume the API.

Table C.2-7: Obtain Access Token request (schema, then example content)

```
HTTP POST {apiRoot}/capif-security/v1/trustedInvokers/{apiInvokerId}/token

Request body:
{
  "grantType": "string",
  "clientId": "string",
  "scope": "string"
}
```

The fields necessary to request the access token are:

- `grantType`: It is used to specify whether the token request is made by the API Invoker themselves (`client_credentials`) or for the use of RNAA (`authorization_code`)
- `clientId`: Identifier of the invoker that is requesting the token.
- `scope`: a String that contains a list of AEF identifiers and its associated API names for which the access token is authorized for use. It takes the format of: 3gpp#aefId1:apiName1,apiName2;aefId2:apiName1,apiName2

The request to obtain the OAuth token in OpenCAPIF is displayed in the following table:

Table C.2-7-OCF: Obtain Access Token request (OCF example)

```
HTTP POST https://capif.mobilesandbox.cloud:37211/capif-
security/v1/trustedInvokers/INVaa03f9911db37be051b243ce3caad9/token


Request body:
{
  "grantType": "client_credentials",
  "clientId": "INVaa03f9911db37be051b243ce3caad9",
  "scope": "3gpp#AEFb1a522967511fd6d6def0cbb3b5f05:/nef/api/v1/3gpp-as-session-with-
qos"
}
```

8. 200 OK

In response, the API Invoker will obtain the token that is needed to consume the API that was specified in the request.

Table C.2-8: Obtain Access Token response (schema)

```
200

Response body:
{
  "access_token": "string",
  "token_type": "string",
  "expires_in": "integer",
  "scope": "string"
}
```

The fields returned in the response are:

- `access_token:` OAuth access token necessary to make the request to the selected API.
- `token_type:` Contains the type of token that is returned (i.e. "Bearer").
- `expires_in:` Contains the number of seconds after which the `access_token` expires.
- `scope:` It is the scope requested in the request and that uses the token to enable the use of an API. It takes the following format: 3gpp#aefId1:apiName1,apiName2;aefId2:apiName1,apiName2

Table C.2-8-OCF: Obtain Access Token response (OCF example)

```
200

Response body:
{
    "access_token": "eyJhbGciOiJSUzI1NiIsInR5cCI6IkpXVCJ9",
    "token_type": "Bearer",
    "expires_in": 600,
    "scope": "3gpp#AEFb1a522967511fd6d6def0cbb3b5f05:3gpp-as-session-with-qos"
}
```

9. CAPIF_API_Invocation

The API invoker's request to the consuming API does not go through CAPIF, rather it is made directly to the API Exposer. For this request, the data obtained from AEF Profile in the Discover request is used (IP/hostname, port, endpoint, method, etc...), it is also necessary to use the access token obtained from CAPIF that will be in the header of the request.

As an example, the request is to the API that has been published as a NEF provider and the OAuth token that has been obtained as an API Invoker.

Table C.2-9: Invoke API request (schema)

```
HTTP {operation} {ipv4Addr}:{port}{uri}

Request headers:

Authorization: "string"
Content-Type: "string"

Request body:

{

  API BODY

}
```

To make the request to the API, the access token obtained from the CAPIF Core Function must be entered in the Authorization header, from there, everything depends on the description of the published API.

Each description contains either the `ipv4Addr` or the `domainName`, both along with the port to which the request can be made. Also, depending on the resource to be called, the appropriate `operation` and `uri` specific to that resource. This is in addition to having to pass data and use the format according to the `dataFormat` field and also specifying this format in the header.

The following example shows how an invoker would use the NEF API that has been published:

Table C.2-9-OCF: Invoke API request (OCF example)

```
HTTP POST https://127.0.0.1:4443/nef/api/3gpp-as-session-with-
qos/v1/{scsAsId}/subscriptions

Request headers:
Authorization: Bearer eyJhbGciOiJSUzI1NiIsInR5cCI6IkpXVCJ9
Content-Type: "application/json"

Request body:
{
    "ipv4Addr": "10.0.0.3",
    "notificationDestination": "http://localhost:80/api/v1/utils/session-with-
qos/callback",
    "snssai": {
        "sst": 1,
        "sd": "000001"
    },
    "dnn": "province1.mnc01.mcc202.gprs",
    "qosReference": 9,
    "altQoSReferences": [0],
    "qosMonInfo": {
        "reqQosMonParams": ["DOWNLINK"],
        "repFreqs": ["EVENT_TRIGGERED"],
        "repThreshDl": 20,
        "repThreshUl": 20,
        "repThreshRp": 50,
        "waitTime": 2,
        "repPeriod": 3
    }
}
```

This would make a request to the NEF API that was used as an example. In this case, the API Invoker would receive the code 201 Created as it is used to create a subscription to the AsSessionWithQoS.

10. CAPIF_API_Invoker_Management (Deregistration)

The invoker can request its deletion from CAPIF using the offboarding service. By using this service, all information stored in the CAPIF Core Function related to the invoker will be deleted, including all the security contexts created for the APIs, ACLs, events subscriptions, etc.

Table C.2-10: DELETE API Invoker request (schema, then example content)

```
HTTP DELETE {apiRoot}/api-invoker-management/v1/onboardedInvokers/{apiInvokerId}
```

For the process of deleting an invoker in CAPIF, the certificate and identifier of the invoker to be deleted is required. That information is obtained through the invoker creation process. Then by making the following request, the invoker is eliminated.

Table C.2-10-OCF: DELETE API Invoker request (OCF example)

```
HTTP DELETE https://capif.mobilesandbox.cloud:37211/api-invoker-
management/v1/onboardedInvokers/INVaa03f9911db37be051b243ce3caad9
```

11. 204 No Content

The successful response will be a 204 No Content, which indicates that the invoker has been successfully deleted along with everything related to it: security context of the APIs; ACLs of the services; events subscriptions etc.

Annex D: CAPIF Test cases

D.1 General

This Annex contains CAPIF test cases including the expected responses from CCF. The test cases defined enable testing not only the success responses (200, 201, 204) but also error codes as defined in 3GPP TS 29.222 [3] Annex A (normative) OpenAPI specification. These test cases are aligned with 3GPP TS 29.222 [3].

The annex defines tests for the following CAPIF APIs:

1. CAPIF_API_Invoker_Management_API
2. CAPIF_API_Provider_Management_API
3. CAPIF_Publish_Service_API
4. CAPIF_Discover_Service_API
5. CAPIF_Events_API
6. CAPIF_Security_API
7. CAPIF_Access_Control_Policy_API
8. CAPIF_Logging_API_Invocation_API
9. CAPIF_Auditing_API

D.2 Test Plan for CAPIF API Invoker Management

This section list test cases for the CAPIF_API_Invoker_Management_API.

Test Case 1: Onboard Invoker

Test ID	capif_api_invoker_management-1
Description	This test will try to register a new Invoker at CAPIF Core
Pre-conditions	The Administrator must have previously registered the User The user must have the access token, the ca root certificate and the URLs to onboard an Invoker. Invoker was not onboarded previously
Execution Steps	1. Onboard Invoker at CCF 2. Store signed Certificate
Information of Test	1. Create CSR, public and private key at invoker 2. Onboard Invoker: a. Send **POST** to https://{CAPIF_HOSTNAME}/api-invoker-management/v1/onboardedInvokers b. "onboardingInformation" -> "apiInvokerPublicKey": must contain CSR generated by Invoker c. Send in Authorization Header the Bearer access_token obtained previously (Authorization:Bearer ${access_token})
Expected Result	1. Onboarding response: a. **Status code 201 Created** b. Response Body must follow **APIInvokerEnrolmentDetails** data structure with: • apiInvokerId • onboardingInformation->apiInvokerCertificate must contain the certificate signed. c. Response Header **Location** must be received with URI to new resource created, following this structure: **{apiRoot}/api-invoker-management/{apiVersion}/onboardedInvokers/{onboardingId}**

Test Case 2: Onboard Invoker Already onboarded

Test ID	capif_api_invoker_management-2
Description	This test will check second onboard of same Invoker is not allowed.
Pre-conditions	The Administrator must have previously registered the User The user must have the access token, the ca root certificate and the URLs to onboard an Invoker. Invoker was onboarded previously
Execution Steps	1. Onboard Invoker at CCF 2. Store signed Certificate 3. Onboard again the Invoker at CCF
Information of Test	1. Perform Onboard Invoker 2. Repeat Onboard Invoker: a. Send **POST** to https://{CAPIF_HOSTNAME}/api-invoker-management/v1/onboardedInvokers b. "onboardingInformation" -> "apiInvokerPublicKey": must contain CSR generated by Invoker c. Send in Authorization Header the Bearer access_token obtained previously (Authorization:Bearer ${access_token})
Expected Result	1. Response to first Onboard request must accomplish: a. **Status code 201 Created** b. Response Body must follow **APIInvokerEnrolmentDetails** data structure with: • apiInvokerId • onboardingInformation->apiInvokerCertificate must contain the certificate signed. c. Response Header **Location** must be received with URI to new resource created, following this structure: **{apiRoot}/api-invoker-management/{apiVersion}/onboardedInvokers/{onboardingId}** **2.** Response to Second Onboard must accomplish: **a.** **Status code 403 Forbidden** **b.** Error Response Body must accomplish with ProblemDetails data structure with: • status 403 • title with message "Forbidden" • detail with message "Invoker Already registered" • cause with message "Identical invoker public key"

Test Case 3: Update Onboarded Invoker

Test ID	capif_api_invoker_management-3
Description	This test will try to update information of previous onboard Invoker at CAPIF Core.
Pre-conditions	The Administrator must have previously registered the User The user must have the access token, the ca root certificate and the URLs to onboard an Invoker. Invoker was onboarded previously with {onboardingId}
Execution Steps	1. Onboard Invoker at CCF 2. Store signed Certificate 3. Update Onboarding Information at CCF with a minor change on "notificationDestination"
Information of Test	1. Perform Onboard Invoker 2. Update information of previously onboarded Invoker: a. Send **PUT** to https://{CAPIF_HOSTNAME}/api-invoker-management/v1/onboardedInvokers/{onboardingId} b. "notificationDestination": "http://host.docker.internal:8086/netapp_new_callback"

Expected Result	1. Response to first Onboard request must accomplish: a. **Status code 201 Created** b. Response Body must follow **APIInvokerEnrolmentDetails** data structure with: • apiInvokerId • onboardingInformation->apiInvokerCertificate must contain the certificate signed. c. Response Header **Location** must be received with URI to new resource created, following this structure: **{apiRoot}/api-invoker-management/{apiVersion}/onboardedInvokers/{onboardingId}** 2. Response to Update Request (PUT) with minor change must contain: **a.** **Status code 200 OK** **b.** "notificationDestination" on response must contain the new value

Test Case 4: Update Not Onboarded Invoker

Test ID	capif_api_invoker_management-4
Description	This test will try to update information of not onboarded Invoker at CAPIF Core.
Pre-conditions	The Administrator must have previously registered the User The user must have the access token, the ca root certificate and the URLs to onboard an Invoker. Invoker was not onboarded previously
Execution Steps	1. Onboard Invoker at CCF 2. Store signed Certificate 3. Update Onboarding Information at CCF of not onboarded
Information of Test	1. Perform Onboard Invoker 2. Update information of not onboarded Invoker: a. Send **PUT** to https://{CAPIF_HOSTNAME}/api-invoker-management/v1/onboardedInvokers/{onboardingId}
Expected Result	1. Response to first Onboard request must accomplish: a. **Status code 201 Created** b. Response Body must follow **APIInvokerEnrolmentDetails** data structure with: • apiInvokerId • onboardingInformation->apiInvokerCertificate must contain the certificate signed. c. Response Header **Location** must be received with URI to new resource created, following this structure: **{apiRoot}/api-invoker-management/{apiVersion}/onboardedInvokers/{onboardingId}** 2. Response to Update Request (PUT) must contain: a. **Status code 404 Not Found** b. Error Response Body must accomplish with ProblemDetails data structure with: • status 404 • title with message "Not Found" • detail with message "Please provide an existing Network App ID" • cause with message "Not exist Network App ID"

Test Case 5: Offboard Invoker

Test ID	capif_api_invoker_management-5
Description	This test case will check that a Registered Invoker can be deleted.
Pre-conditions	The Administrator must have previously registered the User The user must have the access token, the ca root certificate and the URLs to onboard an Invoker. Invoker was onboarded previously

Execution Steps	1. Onboard Invoker at CCF 2. Store signed Certificate 3. Offboard Invoker at CCF
Information of Test	1. Perform Onboard Invoker 2. Offboard onboarded Invoker: a. Send DELETE to https://{CAPIF_HOSTNAME}/api-invoker-management/v1/onboardedInvokers/{onboardingId}
Expected Result	1. Response to first Onboard request must accomplish: a. **Status code 201 Created** b. Response Body must follow **APIInvokerEnrolmentDetails** data structure with: • apiInvokerId • onboardingInformation->apiInvokerCertificate must contain the certificate signed. **c.** Response Header **Location** must be received with URI to new resource created, following this structure: **{apiRoot}/api-invoker-management/{apiVersion}/onboardedInvokers/{onboardingId}** 2. Response to Offboard Request (PUT) must contain: a. **Status code 204 Not Content**

Test Case 6: Offboard Not previously Onboarded Invoker

Test ID	capif_api_invoker_management-6
Description	This test case will check that a Non-Registered Network App cannot be deleted.
Pre-conditions	The Administrator must have previously registered the User The user must have the access token, the ca root certificate and the URLs to onboard an Invoker. Invoker was not onboarded previously
Execution Steps	1. Onboard Invoker at CCF 2. Store signed Certificate 3. Offboard Invoker at CCF
Information of Test	1. Perform Onboard Invoker 2. Offboard onboarded Invoker: a. Send DELETE to https://{CAPIF_HOSTNAME}/api-invoker-management/v1/onboardedInvokers/{onboardingId}
Expected Result	1. Response to first Onboard request must accomplish: a. **Status code 201 Created** b. Response Body must follow **APIInvokerEnrolmentDetails** data structure with: • apiInvokerId • onboardingInformation->apiInvokerCertificate must contain the certificate signed. **c.** Response Header **Location** must be received with URI to new resource created, following this structure: **{apiRoot}/api-invoker-management/{apiVersion}/onboardedInvokers/{onboardingId}** 2. Response to Offboard Request (DELETE) must contain: a. **Status code 404 Not Found** b. Error Response Body must accomplish with ProblemDetails data structure with: • status 404 • title with message "Not Found" • detail with message "Please provide an existing Network App ID" • cause with message "Not exist Network App ID"

Test Case 7: Update Onboarded Invoker Certificate

Test ID	capif_api_invoker_management-7
Description	This test will try to update public key and get a new signed certificate by CAPIF Core.
Pre-conditions	The Administrator must have previously registered the User The user must have the access token, the ca root certificate and the URLs to onboard an Invoker. Invoker was onboarded previously with {onboardingId}
Execution Steps	1. Onboard Invoker at CCF 2. Store signed Certificate 3. Update Onboarding Information at CCF with a new public key and CSR
Information of Test	1. Perform Onboard Invoker 2. Create new CSR, public and private key at invoker 3. Update information of previously onboarded Invoker: a. Send **PUT** to https://{CAPIF_HOSTNAME}/api-invoker-management/v1/onboardedInvokers/{onboardingId} b. Store new invoker certificate.
Expected Result	1. Response to first Onboard request must accomplish: a. **Status code 201 Created** b. Response Body must follow **APIInvokerEnrolmentDetails** data structure with: • apiInvokerId • onboardingInformation->apiInvokerCertificate must contain the certificate signed. **c.** Response Header **Location** must be received with URI to new resource created, following this structure: **{apiRoot}/api-invoker-management/{apiVersion}/onboardedInvokers/{onboardingId}** 2. Response to Update Request (PUT) must contain: **a. Status code 200 OK** **b.** Response Body must follow **APIInvokerEnrolmentDetails** data structure with: • apiInvokerCertificate with new certificate on response -> store to use.

D.3 Test Plan for CAPIF API Provider Management

This section list test cases for the CAPIF_API_Provider_Management_API.

Test Case 1: Onboard API Provider

Test ID	capif_api_provider_management-1
Description	This test case will check that Api Provider can be registered at CCF
Pre-conditions	The Administrator must have previously registered the User The user must have the access token, the ca root certificate and the URLs to onboard a Provider. Provider was not onboarded previously
Execution Steps	1. Onboard Provider at CCF 2. Store signed Certificates
Information of Test	1. Create CSR, public and private key for each function. 2. Onboard Provider: a. Send **POST** to https://{CAPIF_HOSTNAME}/api-provider-management/v1/registrations b. Each function in list “apiProvFuncs” must contain a CSR generated by Provider in “regInfo”-> ”apiProvPubKey”

	c. Send in Authorization Header the Bearer access_token obtained previously (Authorization:Bearer ${access_token})
Expected Result	1. Onboarding response: a. **Status code 201 Created** b. Response Body must follow **APIProviderEnrolmentDetails** data structure. c. For each "apiProvFuncs", we must check: • "apiProvFuncId" is set • "apiProvCert" under "regInfo" is set properly **d.** Response Header **Location** must be received with URI to new resource created, following this structure: **{apiRoot}/api-provider -management/{apiVersion}/registrations /{registrationId}**

Test Case 2: Onboard API Provider Already Onboarded

Test ID	capif_api_provider_management-2
Description	This test case will check that an API Provider previously registered cannot be re-registered
Pre-conditions	The Administrator must have previously registered the User The user must have the access token, the ca root certificate and the URLs to onboard a Provider. Provider was onboarded previously
Execution Steps	1. Onboard Provider at CCF 2. Store signed Certificates 3. Onboard Again the Provider at CCF
Information of Test	1. Create CSR, public and private key for each function. 2. Onboard Provider: a. Send **POST** to https://{CAPIF_HOSTNAME}/api-provider-management/v1/registrations b. Each function in list "apiProvFuncs" must contain a CSR generated by Provider in "regInfo"-> "apiProvPubKey" c. Send in Authorization Header the Bearer access_token obtained previously (Authorization:Bearer ${access_token}) 3. Onboard Again the Provider: a. Send **POST** to **https://{CAPIF_HOSTNAME}/api-provider-management/v1/registrations** b. Each function in list "apiProvFuncs" must contain a CSR generated by Provider in "regInfo"-> "apiProvPubKey" 4. Send in Authorization Header the Bearer access_token obtained previously (Authorization:Bearer ${access_token})
Expected Result	1. Response to first Onboard request must accomplish: a. Response Body must follow **APIProviderEnrolmentDetails** data structure. b. For each "apiProvFuncs", we must check: • "apiProvFuncId" is set • "apiProvCert" under "regInfo" is set properly **c.** Response Header **Location** must be received with URI to new resource created, following this structure: **{apiRoot}/api-provider -management/{apiVersion}/registrations /{registrationId}** **2.** Response to Second Onboard must accomplish: **a. Status code 403 Forbidden** **b.** Error Response Body must accomplish with ProblemDetails data structure with: • status 403 • title with message "Forbidden" • detail with message "Provider Already registered" • cause with message "Identical Provider reg sec"

Test Case 3: Update Onboarded API Provider

Test ID	capif_api_provider_management-3
Description	This test case will check that a Registered API Provider can be updated
Pre-conditions	The Administrator must have previously registered the User The user must have the access token, the ca root certificate and the URLs to onboard a Provider. Provider was onboarded previously
Execution Steps	1. Onboard Provider at CCF 2. Update Provider
Information of Test	1. Create CSR, public and private key for each function. 2. Onboard Provider 3. Update information of previously onboarded Provider: a. Send **PUT** to https://{CAPIF_HOSTNAME}/api-invoker-management/v1/onboardedInvokers/{onboardingId} b. "apiProvDomInfo": "ROBOT_TESTING_MOD" c. Use AMF Certificate
Expected Result	1. Response to first Onboard request must accomplish: a. Response Body must follow **APIProviderEnrolmentDetails** data structure. b. For each "apiProvFuncs", we must check: • "apiProvFuncId" is set • "apiProvCert" under "regInfo" is set properly **c.** Response Header **Location** must be received with URI to new resource created, following this structure: **{apiRoot}/api-provider -management/{apiVersion}/registrations /{registrationId}** **2.** Update Provider must accomplish: **a. Status code 200 OK** **b.** Body returned must accomplish APIProviderEnrolmentDetails data structure, with • "apiProvDomInfo" set to ROBOT_TESTING_MOD

Test Case 4: Update API Provider Not Onboarded

Test ID	capif_api_provider_management-2
Description	This test case will check that a Non-Registered API Provider cannot be updated
Pre-conditions	The Administrator must have previously registered the User The user must have the access token, the ca root certificate and the URLs to onboard a Provider. Provider was not onboarded previously
Execution Steps	1. Onboard Provider at CCF 2. Update Provider
Information of Test	1. Create CSR, public and private key for each function. 2. Onboard Provider 3. Update information of previously onboarded Provider: a. Send **PUT** to https://{CAPIF_HOSTNAME}/api-invoker-management/v1/onboardedInvokers/{onboardingId} b. "apiProvDomInfo": "ROBOT_TESTING_MOD"
Expected Result	1. Response to first Onboard request must accomplish: a. Response Body must follow **APIProviderEnrolmentDetails** data structure. b. For each "apiProvFuncs", we must check: • "apiProvFuncId" is set • "apiProvCert" under "regInfo" is set properly **c.** Response Header **Location** must be received with URI to new resource created, following this structure:

	{apiRoot}/api-provider -management/{apiVersion}/registrations /{registrationId} **2.** Update Provider must accomplish: **a.** **Status code 404 Not Found** **b.** Error Response Body must accomplish with ProblemDetails data structure with: • status 404 • title with message "Not Found" • detail with message "Not Exist Provider Enrolment Details" • cause with message "Not found registrations to Send **THIS** api provider details"

Test Case 5: Partially Update Onboarded API Provider

Test ID	capif_api_provider_management-5
Description	This test case will check that a Registered API Provider can be partially updated
Pre-conditions	The Administrator must have previously registered the User The user must have the access token, the ca root certificate and the URLs to onboard a Provider. Provider was onboarded previously
Execution Steps	1. Onboard Provider at CCF 2. Partial Update Provider
Information of Test	1. Create CSR, public and private key for each function. 2. Onboard Provider 3. Partial update information of previously onboarded Provider: a. Send **PATCH** to https://{CAPIF_HOSTNAME}/api-invoker-management/v1/onboardedInvokers/{onboardingId} b. "apiProvDomInfo": "ROBOT_TESTING_MOD" c. Use AMF Certificate
Expected Result	1. Response to first Onboard request must accomplish: a. Response Body must follow **APIProviderEnrolmentDetails** data structure. b. For each "apiProvFuncs", we must check: • "apiProvFuncId" is set • "apiProvCert" under "regInfo" is set properly **c.** Response Header **Location** must be received with URI to new resource created, following this structure: **{apiRoot}/api-provider -management/{apiVersion}/registrations /{registrationId}** **2.** Partial update Provider must accomplish: **a.** **Status code 200 OK** **b.** Body returned must accomplish APIProviderEnrolmentDetails data structure, with • "apiProvDomInfo" set to ROBOT_TESTING_MOD

Test Case 6: Partially Update API Provider Not Onboarded

Test ID	capif_api_provider_management-6
Description	This test case will check that a Non-Registered API Provider cannot be partially updated
Pre-conditions	The Administrator must have previously registered the User The user must have the access token, the ca root certificate and the URLs to onboard a Provider. Provider was not onboarded previously
Execution Steps	1. Onboard Provider at CCF 2. Partial Update Provider
Information of Test	1. Create CSR, public and private key for each function. 2. Onboard Provider

	3. Partial update information of previously onboarded Provider: 4. Send **PATCH** to https://{CAPIF_HOSTNAME}/api-invoker-management/v1/onboardedInvokers/{onboardingId} 5. "apiProvDomInfo": "ROBOT_TESTING_MOD"
Expected Result	1. Response to first Onboard request must accomplish: a. Response Body must follow **APIProviderEnrolmentDetails** data structure. b. For each "apiProvFuncs", we must check: • "apiProvFuncId" is set • "apiProvCert" under "regInfo" is set properly **c.** Response Header **Location** must be received with URI to new resource created, following this structure: **{apiRoot}/api-provider -management/{apiVersion}/registrations /{registrationId}** **2.** Partial update Provider must accomplish: **a. Status code 404 Not Found** **b.** Error Response Body must accomplish with ProblemDetails data structure with: • status 404 • title with message "Not Found" • detail with message "Not Exist Provider Enrolment Details" • cause with message "Not found registrations to Send **THIS** api provider details"

Test Case 7: Delete Onboarded API Provider

Test ID	capif_api_provider_management-7
Description	This test case will check that a Registered API Provider can be deleted
Pre-conditions	The Administrator must have previously registered the User The user must have the access token, the ca root certificate and the URLs to onboard a Provider. Provider was onboarded previously
Execution Steps	1. Onboard Provider at CCF 2. Store signed Certificates 3. Delete Provider
Information of Test	1. Create CSR, public and private key for each function. 2. Onboard Provider 3. Delete Provider: a. Send **DELETE** to https://{CAPIF_HOSTNAME}/api-provider-management/v1/registrations/{registrationId} b. Use AMF certificate
Expected Result	1. Response to first Onboard request must accomplish: a. Response Body must follow **APIProviderEnrolmentDetails** data structure. b. For each "apiProvFuncs", we must check: • "apiProvFuncId" is set • "apiProvCert" under "regInfo" is set properly **c.** Response Header **Location** must be received with URI to new resource created, following this structure: **{apiRoot}/api-provider -management/{apiVersion}/registrations /{registrationId}** **2.** Delete Provider must accomplish: **a. Status code 204 Not Found**

Test Case 8: Delete not Onboarded API Provider

Test ID	capif_api_provider_management-8
Description	This test case will check that a Non-Registered API Provider cannot be deleted

Pre-conditions	The Administrator must have previously registered the User The user must have the access token, the ca root certificate and the URLs to onboard a Provider. Provider was not onboarded previously
Execution Steps	1. Onboard Provider at CCF 2. Store signed Certificates 3. Delete Provider
Information of Test	1. Create CSR, public and private key for each function. 2. Onboard Provider: 3. Delete Provider: a. Send **DELETE** to https://{CAPIF_HOSTNAME}/api-provider-management/v1/registrations/{registrationId} b. Use AMF certificate
Expected Result	1. Response to first Onboard request must accomplish: a. Response Body must follow **APIProviderEnrolmentDetails** data structure. b. For each “apiProvFuncs”, we must check: • “apiProvFuncId” is set • “apiProvCert” under “regInfo” is set properly **c.** Response Header **Location** must be received with URI to new resource created, following this structure: **{apiRoot}/api-provider-management/{apiVersion}/registrations /{registrationId}** **2.** Responso to Delete Provider must accomplish: **a. Status code 404 Not Found** **b.** Error Response Body must accomplish with ProblemDetails data structure with: • status 404 • title with message “Not Found” • detail with message “Not Exist Provider Enrolment Details” • cause with message “Not found registrations to Send **THIS** api provider details”

D.4 Test Plan for CAPIF API Publish Service

This section list test cases for the CAPIF_Publish_Service_API.

Test Case 1: Publish API by Authorised API Publisher

Test ID	capif_api_publish_service-1
Description	This test case will check that an API Publisher can Publish an API
Pre-conditions	The Administrator must have previously registered the User The user must have the access token, the ca root certificate and the URLs to onboard a Provider. Provider was onboarded previously
Execution Steps	1. Onboard Provider at CCF 2. Publish Service API 3. Retrieve {apiId} from body and Location header with new resource created from response
Information of Test	1. Onboard Provider 2. Publish Service API: a. Send POST to https://{CAPIF_HOSTNAME}/published-apis/v1/{apfId}/service-apis b. Body serviceAPIDescription with apiName service_1 c. Use **APF Certificate**
Expected Result	1. Response to Publish request must accomplish: a. Status code **201 Created** b. Response body must follow ServiceAPIDescription data structure with: • apiId c. Response Header Location must be received with URI to new resource created, following this structure: **{apiRoot}/published-apis/{apiVersion}/{apfId}/service-apis/{serviceApiId}** 2. Published Service API is stored in CAPIF Database.

Test Case 2: Publish API by NON-Authorised API Publisher

Test ID	capif_api_publish_service-2
Description	This test case will check that an API Publisher cannot Publish an API without valid apfId
Pre-conditions	The Administrator must have previously registered the User The user must have the access token, the ca root certificate and the URLs to onboard a Provider. Provider was not onboarded
Execution Steps	1. Onboard Provider at CCF 2. Publish Service API with invalid APF ID
Information of Test	1. Onboard Provider 2. Publish Service API with invalid ID at CCF: a. Send POST to https://{CAPIF_HOSTNAME}/published-apis/v1/{apfIdNotValid}/service-apis b. Body serviceAPIDescription with apiName service_1 c. Use **APF Certificate**
Expected Result	1. Response to Publish request must accomplish: a. Status code **401 Unauthorized** b. Error Response Body must accomplish with ProblemDetails data structure with: • Status 401 • title with message “Unauthorized” • detail with message “Publisher not existing” • cause with message “Publisher id not found” 2. Published Service API is **NOT** stored in CAPIF Database.

Test Case 3: Retrieve all APIs Published by Authorised API Publisher

Test ID	capif_api_publish_service-3
Description	This test case will check that an API Publisher can Retrieve all API published
Pre-conditions	The Administrator must have previously registered the User The user must have the access token, the ca root certificate and the URLs to onboard a Provider. Provider was onboarded previously At least 2 services APIs are published
Execution Steps	1. Onboard Provider at CCF 2. Publish Service API service_1 3. Retrieve {apiId1} from body and Location header with new resource created from response 4. Publish Service API service_2 5. Retrieve {apiId2} from body and Location header with new resource created from response 6. Retrieve All published APIs and check if both are present
Information of Test	1. Onboard Provider 2. Publish Service API at CCF: a. Send POST to https://{CAPIF_HOSTNAME}/published-apis/v1/{apfId}/service-apis b. Body serviceAPIDescription with apiName service_1 c. Get apiId1 d. Use **APF Certificate** 3. Publish Other Service API at CCF: a. Send POST to https://{CAPIF_HOSTNAME}/published-apis/v1/{apfId}/service-apis b. Body serviceAPIDescription with apiName service_2 c. Get apiId2 d. Use **APF Certificate** 4. Retrieve all published APIs: a. Send GET to https://{CAPIF_HOSTNAME}/published-apis/v1/{apfId}/service-apis

	b. Use APF Certificate
Expected Result	1. Response to service 1 Publish request must accomplish: a. Status code **201 Created** b. Response body must follow ServiceAPIDescription data structure with: • apiId1 c. Response Header Location must be received with URI to new resource created, following this structure: **{apiRoot}/published-apis/{apiVersion}/{apfId}/service-apis/{serviceApiId1}** 2. Response to service 2 Publish request must accomplish: a. Status code **201 Created** b. Response body must follow ServiceAPIDescription data structure with: • apiId2 c. Response Header Location must be received with URI to new resource created, following this structure: **{apiRoot}/published-apis/{apiVersion}/{apfId}/service-apis/{serviceApiId2}** 3. Published Service APIs are stored in CAPIF Database 4. Response to Retrieve all published APIs a. 200 OK b. Response body must return an array of ServiceAPIDescription data. c. Array must contain all previously published APIs.

Test Case 4: Retrieve all APIs Published by NON Authorised API Publisher

Test ID	capif_api_publish_service-4
Description	This test case will check that an API Publisher cannot Retrieve API published when apfId is not authorised
Pre-conditions	The Administrator must have previously registered the User The user must have the access token, the ca root certificate and the URLs to onboard a Provider. Provider was not onboarded previously At least 2 services APIs are published
Execution Steps	1. Onboard Provider at CCF 2. Retrieve All published APIs
Information of Test	1. Onboard Provider 2. Retrieve all published APIs: a. Send GET to https://{CAPIF_HOSTNAME}/published-apis/v1/{apfId}/service-apis b. Use APF Certificate
Expected Result	1. Response to Publish request must accomplish: a. Status code **401 Unauthorized** b. Error Response Body must accomplish with ProblemDetails data structure with: • Status 401 • title with message "Unauthorized" • detail with message "Publisher not existing" • cause with message "Publisher id not found" 2. Published Service API is **NOT** stored in CAPIF Database.

Test Case 5: Retrieve single APIs Published by Authorised API Publisher

Test ID	capif_api_publish_service-5
Description	This test case will check that an API Publisher can Retrieve API published one by one
Pre-conditions	The Administrator must have previously registered the User The user must have the access token, the ca root certificate and the URLs to onboard a Provider.

	Provider was onboarded previously At least 2 services APIs are published
Execution Steps	1. Onboard Provider at CCF 2. Publish Service API service_1 3. Retrieve {apiId1} from body and Location header with new resource created from response 4. Publish Service API service_2 5. Retrieve {apiId2} from body and Location header with new resource created from response 6. Retrieve service_1 API Detail 7. Retrieve service_2 API Detail
Information of Test	1. Onboard Provider 2. Publish Service API at CCF: a. Send POST to https://{CAPIF_HOSTNAME}/published-apis/v1/{apfId}/service-apis b. Body serviceAPIDescription with apiName service_1 c. Get apiId1 d. Use APF Certificate 3. Publish Other Service API at CCF: a. Send POST to https://{CAPIF_HOSTNAME}/published-apis/v1/{apfId}/service-apis b. Body serviceAPIDescription with apiName service_2 c. Get apiId2 d. Use APF Certificate 4. Retrieve service_1 published API Details: a. Send GET to https://{CAPIF_HOSTNAME}/published-apis/v1/{apfId}/service-apis/{apiId1} b. Use APF Certificate 5. Retrieve service_2 published API Details: a. Send GET to https://{CAPIF_HOSTNAME}/published-apis/v1/{apfId}/service-apis/{apiId2} b. Use APF Certificate
Expected Result	1. Response to service_1 Publish request must accomplish: a. Status code **201 Created** b. Response body must follow ServiceAPIDescription data structure with: • apiId1 c. Response Header Location must be received with URI to new resource created, following this structure: **{apiRoot}/published-apis/{apiVersion}/{apfId}/service-apis/{serviceApiId1}** 2. Response to service_2 Publish request must accomplish: a. Status code **201 Created** b. Response body must follow ServiceAPIDescription data structure with: • apiId2 c. Response Header Location must be received with URI to new resource created, following this structure: **{apiRoot}/published-apis/{apiVersion}/{apfId}/service-apis/{serviceApiId2}** 3. Published Service APIs are stored in CAPIF Database 4. Response to Retrieve service_1 published API using apiId1: a. 200 OK b. Response body must return an array of ServiceAPIDescription data. c. Array must contain same information than service_1 published registration response. 5. Response to Retrieve service_2 published API using apiId2: a. 200 OK b. Response body must return an array of ServiceAPIDescription data. c. Array must contain same information than service_2 published registration response.

Test Case 6: Retrieve single APIs NON Published by Authorised API Publisher

Test ID	capif_api_publish_service-6
Description	This test case will check that an API Publisher try to get detail of not published API.

Pre-conditions	The Administrator must have previously registered the User The user must have the access token, the ca root certificate and the URLs to onboard a Provider. Provider was onboarded previously No published API.
Execution Steps	1. Onboard Provider at CCF 2. Retrieve not published API Detail
Information of Test	1. Onboard Provider 2. Retrieve not published API Details: a. Send GET to https://{CAPIF_HOSTNAME}/published-apis/v1/{apfId}/service-apis/{SERVICE_API_ID_NOT_VALID} b. Use APF Certificate
Expected Result	1. Response to Retrieve for NOT published API must accomplish: a. Status code **404 Not Found** b. Error Response Body must accomplish with ProblemDetails data structure with: • Status 404 • title with message “Not Found” • detail with message “Service API not found” • cause with message “No Service with specific credentials exists”

Test Case 7: Retrieve single APIs Published by NON Authorised API Publisher

Test ID	capif_api_publish_service-7
Description	This test case will check that an API Publisher cannot Retrieve detailed API published when apfId is not authorised
Pre-conditions	The Administrator must have previously registered the User The user must have the access token, the ca root certificate and the URLs to onboard a Provider. The user must have the access token, the ca root certificate and the URLs to onboard an Invoker. Provider was onboarded Invoker was onboarded
Execution Steps	1. Onboard Provider at CCF 2. Publish Service API at CCF 3. Retrieve {apiId1} from body and Location header with new resource created from response 4. Onboard Invoker at CCF and store signed Invoker Certificate 5. Retrieve detailed published API acting as Invoker
Information of Test	1. Onboard Provider and Invoker 2. Publish Service API at CCF: a. Send POST to https://{CAPIF_HOSTNAME}/published-apis/v1/{apfId}/service-apis b. Body serviceAPIDescription with apiName service_1 c. Get apiId1 d. Use APF Certificate 3. Retrieve service_1 published API Details: a. Send GET to https://{CAPIF_HOSTNAME}/published-apis/v1/{apfId}/service-apis/{apiId1} b. Use Invoker Certificate
Expected Result	1. Response to Retrieve for NOT published API must accomplish: a. Status code **401 Unauthorized** b. Error Response Body must accomplish with ProblemDetails data structure with: • Status 401 • title with message “Unauthorized” • detail with message “User not authorized” • cause with message “Certificate not authorized”

Test Case 8: Update API Published by Authorised API Publisher with valid serviceApiId

Test ID	capif_api_publish_service-8
Description	This test case will check that an API Publisher can Update published API with a valid serviceApiId
Pre-conditions	The Administrator must have previously registered the User The user must have the access token, the ca root certificate and the URLs to onboard a Provider. Provider was onboarded previously A service APIs is published
Execution Steps	1. Onboard Provider at CCF 2. Publish Service API 3. Retrieve {apiId} from body and Location header with new resource created from response 4. Update published Service API 5. Retrieve detail of Service API
Information of Test	1. Onboard Provider 2. Publish Service API at CCF: a. Send POST to **https://{CAPIF_HOSTNAME}/published-apis/v1/{apfId}/service-apis** b. Body serviceAPIDescription with apiName service_1 c. Get apiId d. Use **APF Certificate** 3. Update published API at CCF: a. Send PUT to **https://{CAPIF_HOSTNAME}/published-apis/v1/{apfId}/service-apis/{apiId}** b. Body serviceAPIDescription with overrided apiName to service_1_modified c. Use **APF Certificate** 4. Retrieve service_1 published API Details: a. Send GET to **https://{CAPIF_HOSTNAME}/published-apis/v1/{apfId}/service-apis/{apiId}** b. Use APF Certificate c. Check apiName is service_1_modified
Expected Result	1. Response to Publish request must accomplish: a. Status code **201 Created** b. Response body must follow ServiceAPIDescription data structure with: • apiId c. Response Header Location must be received with URI to new resource created, following this structure: **{apiRoot}/published-apis/{apiVersion}/{apfId}/service-apis/{serviceApiId}** 2. Response to Update Published service request must accomplish: a. Status code **200 OK** b. Response body must follow ServiceAPIDescription data structure with: • apiName service_1_modified 3. Response to Retrieve detail of Service API: a. 200 OK b. Response body must follow ServiceAPIDescription data structure with: • apiName service_1_modified

Test Case 9: Update API Published by Authorised API Publisher with invalid serviceApiId

Test ID	capif_api_publish_service-9
Description	This test case will check that an API Publisher cannot Update published API with a invalid serviceApiId

Pre-conditions	The Administrator must have previously registered the User The user must have the access token, the ca root certificate and the URLs to onboard a Provider. Provider was onboarded previously A service API is published
Execution Steps	1. Onboard Provider at CCF 2. Publish Service API 3. Retrieve {apiId} from body and Location header with new resource created from response 4. Update published Service API

Information of Test	1. Onboard Provider
	2. Publish Service API at CCF:
	  a. Send POST to **https://{CAPIF_HOSTNAME}/published-apis/v1/{apfId}/service-apis**
	  b. Body serviceAPIDescription with apiName service_1
	  c. Get apiId
	  d. Use **APF Certificate**
	3. Update published API at CCF:
	  a. Send PUT to **https://{CAPIF_HOSTNAME}/published-apis/v1/{apfId}/service-apis/{SERVICE_API_ID_NOT_VALID}**
	  b. Body serviceAPIDescription with overrided apiName to service_1_modified
	  c. Use **APF Certificate**

Expected Result	1. Response to Publish request must accomplish: a. Status code **201 Created** b. Response body must follow ServiceAPIDescription data structure with: • apiId c. Response Header Location must be received with URI to new resource created, following this structure: **{apiRoot}/published-apis/{apiVersion}/{apfId}/service-apis/{serviceApiId}** 2. Response to Update Published service request must accomplish: a. Status code 4**04 Not Found** b. Error Response Body must accomplish with ProblemDetails data structure with: • Status 404 • title with message "Not Found" • detail with message "Service API not found"

Test Case 10: Update API Published by NON Authorised API Publisher

Test ID	capif_api_publish_service-10
Description	This test case will check that an API Publisher cannot Update API published when apfId is not authorised
Pre-conditions	The Administrator must have previously registered the User The user must have the access token, the ca root certificate and the URLs to onboard a Provider. The user must have the access token, the ca root certificate and the URLs to onboard an Invoker. Provider was onboarded Invoker was onboarded
Execution Steps	1. Onboard Provider at CCF 2. Publish Service API at CCF 3. Retrieve {apiId1} from body and Location header with new resource created from response 4. Onboard Invoker at CCF and store signed Invoker Certificate 5. Update published API at CCF as Invoker 6. Retrieve detail of Service API as Provider

Information of Test	1. Onboard Provider and Invoker
	2. Publish Service API at CCF:
	  a. Send POST to **https://{CAPIF_HOSTNAME}/published-apis/v1/{apfId}/service-apis**
	  b. Body serviceAPIDescription with apiName service_1
	  c. Get apiId

	d. Use **APF Certificate** 3. Update published API at CCF: a. Send PUT to **https://{CAPIF_HOSTNAME}/published-apis/v1/{apfId}/service-apis/{SERVICE_API_ID_NOT_VALID}** b. Body serviceAPIDescription with overrided apiName to service_1_modified c. Use **Invoker Certificate** 4. Retrieve service_1 published API Details: a. Send GET to **https://{CAPIF_HOSTNAME}/published-apis/v1/{apfId}/service-apis/{apiId}** b. Use APF Certificate c. Check apiName is service_1
Expected Result	1. Response to Update published API acting as Invoker must accomplish: a. Status code **401 Unauthorized** b. Error Response Body must accomplish with ProblemDetails data structure with: • Status 401 • title with message "Unauthorized" • detail with message "User not authorized" • cause with message "Certificate not authorized" 2. Response to Retrieve Detail of Service API a. 200 OK b. Response Body must follow ServiceAPIDescription data structure with: • apiName service_1

Test Case 11: Delete API Published by Authorised API Publisher with valid serviceApiId

Test ID	capif_api_publish_service-11
Description	This test case will check that an API Publisher can Delete published API with a valid serviceApiId
Pre-conditions	The Administrator must have previously registered the User The user must have the access token, the ca root certificate and the URLs to onboard a Provider. Provider was onboarded previously A service API is published
Execution Steps	1. Onboard Provider at CCF 2. Publish Service API 3. Retrieve {apiId} from body and Location header with new resource created from response 4. Remove published Service API at CCF 5. Try to retrieve deleted Service API from CCF
Information of Test	1. Onboard Provider 2. Publish Service API at CCF: a. Send POST to **https://{CAPIF_HOSTNAME}/published-apis/v1/{apfId}/service-apis** b. Body serviceAPIDescription with apiName service_1 c. Get apiId d. Use **APF Certificate** 3. Remove published API at CCF: a. Send DELETE to **https://{CAPIF_HOSTNAME}/published-apis/v1/{apfId}/service-apis/{apiId}** b. Use **APF Certificate** 4. Retrieve service_1 published API Details: a. Send GET to **https://{CAPIF_HOSTNAME}/published-apis/v1/{apfId}/service-apis/{apiId}** b. Use APF Certificate
Expected Result	1. Response to Publish request must accomplish: a. Status code **201 Created** b. Response body must follow ServiceAPIDescription data structure with: • apiId

	c. Response Header Location must be received with URI to new resource created, following this structure: **{apiRoot}/published-apis/{apiVersion}/{apfId}/service-apis/{serviceApiId}** 2. Response to Remove Published service request must accomplish: a. Status code **204 NoContent** 3. Response to Retrieve detail for DELETED published API must accomplish: a. Status code 4**04 Not Found** b. Error Response Body must accomplish with ProblemDetails data structure with: • Status 404 • title with message "Service API not found" • detail with message "No Service with specific credentials exists"

Test Case 12: Delete API Published by Authorised API Publisher with invalid serviceApiId

Test ID	capif_api_publish_service-12
Description	This test case will check that an API Publisher cannot Delete with invalid serviceApiId
Pre-conditions	The Administrator must have previously registered the User The user must have the access token, the ca root certificate and the URLs to onboard a Provider. Provider was onboarded previously A service API is published
Execution Steps	1. Onboard Provider at CCF 2. Publish Service API 3. Retrieve {apiId} from body and Location header with new resource created from response 4. Remove published Service API at CCF
Information of Test	1. Onboard Provider 2. Publish Service API at CCF: a. Send POST to **https://{CAPIF_HOSTNAME}/published-apis/v1/{apfId}/service-apis** b. Body serviceAPIDescription with apiName service_1 c. Get apiId d. Use **APF Certificate** 3. Remove published API at CCF: a. Send DELETE to **https://{CAPIF_HOSTNAME}/published-apis/v1/{apfId}/service-apis/{SERVICE_API_ID_NOT_VALID}** b. Use **APF Certificate**
Expected Result	1. Response to Publish request must accomplish: a. Status code **201 Created** b. Response body must follow ServiceAPIDescription data structure with: • apiId c. Response Header Location must be received with URI to new resource created, following this structure: **{apiRoot}/published-apis/{apiVersion}/{apfId}/service-apis/{serviceApiId}** 2. Response to Remove Published service request must accomplish: a. Status code 4**04 Not Found** b. Error Response Body must accomplish with ProblemDetails data structure with: • Status 404 • Title with message "Not Found" • title with message "Service API not found" • detail with message "Service API id not found"

Test Case 13: Delete API Published by NON Authorised API Publisher with valid serviceApiId

Test ID	capif_api_publish_service-12
Description	This test case will check that an API Publisher cannot Delete with invalid serviceApiId
Pre-conditions	The Administrator must have previously registered the User The user must have the access token, the ca root certificate and the URLs to onboard a Provider. Provider was onboarded previously A service APIs is published
Execution Steps	1. Onboard Provider at CCF 2. Publish Service API 3. Retrieve {apiId} from body and Location header with new resource created from response 4. Onboard Invoker at CCF and store signed Invoker Certificate 5. Remove published Service API at CCF with invalid serviced as Invoker
Information of Test	1. Onboard Provider and Invoker 2. Publish Service API at CCF: a. Send POST to **https://{CAPIF_HOSTNAME}/published-apis/v1/{apfId}/service-apis** b. Body serviceAPIDescription with apiName service_1 c. Get apiId d. Use **APF Certificate** 3. Remove published API at CCF as Invoker: a. Send DELETE to **https://{CAPIF_HOSTNAME}/published-apis/v1/{apfId}/service-apis/{apiId}** b. Use **Invoker Certificate**
Expected Result	1. Response to Remove Published service request must accomplish: a. Status code **401 Unauthorized** b. Error Response Body must accomplish with ProblemDetails data structure with: • Status 401 • title with message “Unauthorized” • detail with message “User not authorized” • cause with message “Certificate not authorized”

D.5 Test Plan for CAPIF Discover Service API

This section list test cases for the CAPIF_Discover_Service_API.

Test Case 1: Discover Published Service Apis by Authorised Invoker

Test ID	capif_api_discover _service-1
Description	This test case will check if Invoker can discover published service APIs.
Pre-conditions	The Administrator must have previously registered the User The user must have the access token, the ca root certificate and the URLs to onboard an Invoker. Service APIs are published
Execution Steps	1. Onboard Provider at CCF and publish Service API at CCF. 2. Onboard Invoker at CCF 3. Discover Service APIs by Invoker
Information of Test	1. Onboard Provider and Invoker 2. Publish Service API: a. Send POST to **https://{CAPIF_HOSTNAME}/published-apis/v1/{apfId}/service-apis** b. Body serviceAPIDescription with apiName service_1

	c. Use **APF Certificate** 3. Request Discover Published APIs: a. Send GET to **https://{CAPIF_HOSTNAME}/service-apis/v1/allServiceAPIs?**api-invoker-id={apiInvokerId} b. Param api-invoker-id is mandatory c. Use Invoker **Certificate**
Expected Result	1. Response to Publish request must accomplish: a. Status code **201 Created** b. Response body must follow ServiceAPIDescription data structure with: • apiId c. Response Header Location must be received with URI to new resource created, following this structure: **{apiRoot}/published-apis/{apiVersion}/{apfId}/service-apis/{serviceApiId}** 2. Response to Discover Request by Invoker: a. Status code 200 OK b. Response body must follow DiscoveredAPIs data structure: • Check if DiscoveredAPIs contains the API Published previously

Test Case 2: Discover Published Service APIs by Non Authorised Invoker

Test ID	capif_api_discover _service-2
Description	This test case will check that an API Publisher can't discover published APIs because is not authorized.
Pre-conditions	The Administrator must have previously registered the User The user must have the access token, the ca root certificate and the URLs to onboard an Invoker. Service APIs are published
Execution Steps	1. Onboard Provider at CCF and publish Service API at CCF. 2. Onboard Invoker at CCF 3. Discover Service APIs by Invoker
Information of Test	1. Onboard Provider and Invoker 2. Publish Service API: a. Send POST to **https://{CAPIF_HOSTNAME}/published-apis/v1/{apfId}/service-apis** b. Body serviceAPIDescription with apiName service_1 c. Use **APF Certificate** 3. Request Discover Published APIs: a. Send GET to **https://{CAPIF_HOSTNAME}/service-apis/v1/allServiceAPIs?**api-invoker-id={apiInvokerId} b. Param api-invoker-id is mandatory c. Use not Invoker **Certificate**
Expected Result	1. Response to Publish request must accomplish: a. Status code **201 Created** b. Response body must follow ServiceAPIDescription data structure with: • apiId c. Response Header Location must be received with URI to new resource created, following this structure: **{apiRoot}/published-apis/{apiVersion}/{apfId}/service-apis/{serviceApiId}** 2. Response to Discover Request by no Invoker: a. Status code **401 Unauthorized** b. Error Response Body must accomplish with ProblemDetails data structure with: • Status 401 • title with message "Unauthorized" • detail with message "User not authorized" • cause with message "Certificate not authorized"

Test Case 3: Discover Published Service APIs by Not Registered API Invoker

Test ID	capif_api_discover _service-3
Description	This test case will check that a not registered invoker is forbidden to discover published APIs.
Pre-conditions	The Administrator must have previously registered the User The user must have the access token, the ca root certificate and the URLs to onboard an Invoker. Service APIs are published
Execution Steps	1. Onboard Provider at CCF and publish Service API at CCF. 2. Onboard Invoker at CCF 3. Discover Service APIs by Invoker
Information of Test	1. Onboard Provider and Invoker 2. Publish Service API: a. Send POST to **https://{CAPIF_HOSTNAME}/published-apis/v1/{apfId}/service-apis** b. Body serviceAPIDescription with apiName service_1 c. Use **APF Certificate** 3. Request Discover Published API with not valid apiInvokerId: a. Send GET to **https://{CAPIF_HOSTNAME}/service-apis/v1/allServiceAPIs?**api-invoker-id={NOT_VALID_INVOKER_ID} b. Param api-invoker-id is mandatory c. Use Invoker **Certificate**
Expected Result	1. Response to Publish request must accomplish: a. Status code **201 Created** b. Response body must follow ServiceAPIDescription data structure with: • apiId c. Response Header Location must be received with URI to new resource created, following this structure: **{apiRoot}/published-apis/{apiVersion}/{apfId}/service-apis/{serviceApiId}** 2. Response to Discover Request by Invoker: a. Status code **404 Not Found** b. Error Response Body must accomplish with ProblemDetails data structure with: • Status 404 • Title with message “Not Found” • title with message “API Invoker does not exist” • detail with message “API Invoker id not found”

Test Case 4: Discover Published Service APIs by Authorised Invoker with 1 result filtered

Test ID	capif_api_discover _service-4
Description	This test case will check if Invoker can discover published service APIs
Pre-conditions	The Administrator must have previously registered the User The user must have the access token, the ca root certificate and the URLs to onboard an Invoker. At least 2 Service APIs are published
Execution Steps	1. Onboard Provider at CCF and publish Service API service_1 and service_2 at CCF. 2. Onboard Invoker at CCF 3. Discover Service APIs by Invoker 4. Discover filtered by api-name service_1 Service APIs by Invoker
Information of Test	1. Onboard Provider and Invoker 2. Publish Service API: a. Send POST to **https://{CAPIF_HOSTNAME}/published-apis/v1/{apfId}/service-apis** b. Body serviceAPIDescription with apiName service_1 c. Use **APF Certificate**

	3. Publish Service API: a. Send POST to **https://{CAPIF_HOSTNAME}/published-apis/v1/{apfId}/service-apis** b. Body serviceAPIDescription with apiName service_2 c. Use **APF Certificate** 4. Request Discover Published APIs filtering by api-name: a. Send GET to **https://{CAPIF_HOSTNAME}/service-apis/v1/allServiceAPIs?**api-invoker-id={apiInvokerId}**&api-name=service_1** b. Param api-invoker-id is mandatory c. Use Invoker **Certificate** d. Filter by api-name service_1
Expected Result	1. Response to Publish service_1 request must accomplish: a. Status code **201 Created** b. Response body must follow ServiceAPIDescription data structure with: • apiId1 c. Response Header Location must be received with URI to new resource created, following this structure: **{apiRoot}/published-apis/{apiVersion}/{apfId}/service-apis/{serviceApiId1}** 2. Response to Publish service_2 request must accomplish: a. Status code **201 Created** b. Response body must follow ServiceAPIDescription data structure with: • apiId2 c. Response Header Location must be received with URI to new resource created, following this structure: **{apiRoot}/published-apis/{apiVersion}/{apfId}/service-apis/{serviceApiId2}** 3. Response to Discover Request by Invoker: a. Status code 200 OK b. Response body must follow DiscoveredAPIs data structure: • Check if DiscoveredAPIs contains only the API Published with api-name **service_1**

Test Case 5: Discover Published Service APIs by Authorised Invoker with no match

Test ID	capif_api_discover _service-5
Description	This test case will check if Invoker can discover published service APIs.
Pre-conditions	The Administrator must have previously registered the User The user must have the access token, the ca root certificate and the URLs to onboard an Invoker. At least 2 Service APIs are published
Execution Steps	1. Onboard Provider at CCF and publish Service API service_1 and service_2 at CCF. 2. Onboard Invoker at CCF 3. Discover Service APIs by Invoker 4. Discover filtered by api-name not published Service API by Invoker
Information of Test	1. Onboard Provider and Invoker 2. Publish Service API: a. Send POST to **https://{CAPIF_HOSTNAME}/published-apis/v1/{apfId}/service-apis** b. Body serviceAPIDescription with apiName service_1 c. Use **APF Certificate** 3. Publish Service API: a. Send POST to **https://{CAPIF_HOSTNAME}/published-apis/v1/{apfId}/service-apis** b. Body serviceAPIDescription with apiName service_2 c. Use **APF Certificate** 4. Request Discover Published APIs filtering by api-name not published: a. Send GET to **https://{CAPIF_HOSTNAME}/service-apis/v1/allServiceAPIs?**api-invoker-id={apiInvokerId}**&api-name=NOT_VALID_NAME**

	b. Param api-invoker-id is mandatory c. Use Invoker **Certificate** d. Filter by api-name NOT_VALID_NAME
Expected Result	1. Response to Publish service_1 request must accomplish: a. Status code **201 Created** b. Response body must follow ServiceAPIDescription data structure with: • apiId1 c. Response Header Location must be received with URI to new resource created, following this structure: **{apiRoot}/published-apis/{apiVersion}/{apfId}/service-apis/{serviceApiId1}** 2. Response to Publish service_2 request must accomplish: a. Status code **201 Created** b. Response body must follow ServiceAPIDescription data structure with: • apiId2 c. Response Header Location must be received with URI to new resource created, following this structure: **{apiRoot}/published-apis/{apiVersion}/{apfId}/service-apis/{serviceApiId2}** 3. Response to Discover Request by Invoker: a. Status code 200 OK b. Response body must follow DiscoveredAPIs data structure: • Check if DiscoveredAPIs contains the previously registered Service API Published. 4. Response to Discover Request By Invoker: a. Status code 4**04 Not Found** b. Error Response Body must accomplish with ProblemDetails data structure with: • Status 404 • Title with message “Not Found” • title with message “API Invoker{api_invoker_id} has no API Published that accomplish filter conditions” • detail with message “No API message Published accomplish filter conditions”

Test Case 6: Discover Published Service APIs by Authorised Invoker not filtered

Test ID	capif_api_discover _service-6
Description	This test case will check if Invoker can discover published service APIs.
Pre-conditions	The Administrator must have previously registered the User The user must have the access token, the ca root certificate and the URLs to onboard an Invoker. At least 2 Service APIs are published
Execution Steps	1. Onboard Provider at CCF and publish Service API service_1 and service_2 at CCF. 2. Onboard Invoker at CCF 3. Discover Service APIs by Invoker 4. Discover without filter by Invoker
Information of Test	1. Onboard Provider and Invoker 2. Publish Service API: a. Send POST to **https://{CAPIF_HOSTNAME}/published-apis/v1/{apfId}/service-apis** b. Body serviceAPIDescription with apiName service_1 c. Use **APF Certificate** 3. Publish Service API: a. Send POST to **https://{CAPIF_HOSTNAME}/published-apis/v1/{apfId}/service-apis** b. Body serviceAPIDescription with apiName service_2 c. Use **APF Certificate** 4. Request Discover Published APIs filtering by api-name not published: a. Send GET to **https://{CAPIF_HOSTNAME}/service-apis/v1/allServiceAPIs?**api-invoker-id={apiInvokerId}

	b. Param api-invoker-id is mandatory c. Use Invoker **Certificate**
Expected Result	1. Response to Publish service_1 request must accomplish: a. Status code **201 Created** b. Response body must follow ServiceAPIDescription data structure with: • apiId1 c. Response Header Location must be received with URI to new resource created, following this structure: **{apiRoot}/published-apis/{apiVersion}/{apfId}/service-apis/{serviceApiId1}** 2. Response to Publish service_2 request must accomplish: a. Status code **201 Created** b. Response body must follow ServiceAPIDescription data structure with: • apiId2 c. Response Header Location must be received with URI to new resource created, following this structure: **{apiRoot}/published-apis/{apiVersion}/{apfId}/service-apis/{serviceApiId2}** 3. Response to Discover Request by Invoker: a. Status code 200 OK b. Response body must follow DiscoveredAPIs data structure: • Check if DiscoveredAPIs contains the 2 previously registered Service API Published.

D.6 Test Plan for CAPIF API Security Service

This section list test cases for the CAPIF_Security_API.

Test Case 1: Creates a Security Context for an API Invoker

Test ID	capif_api_security-1
Description	This test case will check that an API Invoker can create a Security context
Pre-conditions	The Administrator must have previously registered the User The user must have the access token, the ca root certificate and the URLs to onboard an Invoker.
Execution Steps	1. Onboard Invoker at CCF. 2. Stored signed Certificate 3. Create Security Context
Information of Test	1. Onboard Invoker at CCF. 2. Create Security Context for this Invoker: a. Send **PUT** https://{CAPIF_HOSTNAME}/trustedInvokers/{apiInvokerId} b. Use **Invoker Certificate**
Expected Result	1. Create security context: a. **201 Created** response. b. body returned must accomplish **ServiceSecurity** data structure. c. Location Header must contain the new resource URL **{apiRoot}/capif-security/v1/trustedInvokers/{apiInvokerId}**

Test Case 2: Creates a Security Context for an API Invoker with provider role

Test ID	capif_api_security-2
Description	This test case will check that a Provider cannot create a Security context with valid apiInvokerId.

Pre-conditions	The Administrator must have previously registered the User The user must have the access token, the ca root certificate and the URLs to onboard an Invoker.
Execution Steps	4. Onboard Invoker at CCF. 5. Onboard Provider at CCF 6. Create Security Context using Provider Certificate
Information of Test	1. Onboard Invoker and provider at CCF. 2. Create Security Context for this Invoker: a. Send **PUT** https://{CAPIF_HOSTNAME}/trustedInvokers/{apiInvokerId} b. Use **AEF Certificate**
Expected Result	1. Create security context using provider certificate: a. **401 Unauthorized** response. b. body returned must accomplish **ProblemDetails** data structure, with: • status **401** • title with message "Unauthorized" • detail with message "Role not authorized for this API route". • cause with message "User role must be invoker". 2. No context stored at Database

Test Case 3: Creates a Security Context for an API Invoker with provider role and invalid apiInvokerId

Test ID	capif_api_security-3
Description	This test case will check that a Provider cannot create a Security context with invalid apiInvokerID.
Pre-conditions	The Administrator must have previously registered the User The user must have the access token, the ca root certificate and the URLs to onboard an Invoker.
Execution Steps	1. Onboard Invoker at CCF. 2. Onboard Provider at CCF 3. Create Security Context using Provider Certificate and not valid apiInvokerId
Information of Test	1. Onboard Invoker and provider at CCF. 2. Create Security Context for this Invoker: a. Send **PUT** https://{CAPIF_HOSTNAME}/trustedInvokers/{API_INVOKER_NOT_VALID} b. Use **AEF Certificate**
Expected Result	3. Create security context using provider certificate: a. **401 Unauthorized** response. b. body returned must accomplish **ProblemDetails** data structure, with: • status **401** • title with message "Unauthorized" • detail with message "Role not authorized for this API route". • cause with message "User role must be invoker". 4. No context stored at Database

Test Case 4: Creates a Security Context for an API Invoker with Invoker role and invalid apiInvokerId

Test ID	capif_api_security-4
Description	This test case will check that an Invoker cannot create a Security context with valid apiInvokerId.
Pre-conditions	The Administrator must have previously registered the User The user must have the access token, the ca root certificate and the URLs to onboard an Invoker.

Execution Steps	1. Onboard Invoker at CCF. 2. Create Security Context using invoker Certificate and not valid apiInvokerId
Information of Test	1. Onboard Invoker and provider at CCF. 2. Create Security Context for this Invoker: a. Send **PUT** https://{CAPIF_HOSTNAME}/trustedInvokers/{API_INVOKER_NOT_VALID} b. Use Invoker **Certificate**
Expected Result	1. Create security context using Invoker certificate: a. **404 Not Found** response. b. body returned must accomplish **ProblemDetails** data structure, with: • status **404** • title with message "Not Found" • detail with message "Invoker not found". • cause with message "API Invoker not exists or invalid ID". 2. No context stored at Database

Test Case 5: retrieve the Security Context of an API Invoker

Test ID	capif_api_security-5
Description	This test case will check that a provider can retrieve the Security context of an API Invoker
Pre-conditions	The Administrator must have previously registered the User The user must have the access token, the ca root certificate and the URLs to onboard an Invoker. API Invoker has created a valid Security Context
Execution Steps	1. Onboard Invoker at CCF. 2. Onboard Provider at CCF. 3. Create Security Context using invoker Certificate 4. Retrieve Security Context by Provider
Information of Test	1. Onboard Invoker and provider at CCF. 2. Create Security Context for this Invoker: a. Send **PUT** https://{CAPIF_HOSTNAME}/trustedInvokers/{apiInvokerId} b. Use Invoker **Certificate** 3. Retrieve Security Context of Invoker by Provider: a. Send **GET** https://{CAPIF_HOSTNAME}/trustedInvokers/{apiInvokerId} b. Using **AEF Certificate**
Expected Result	3. Retrieve security context: a. **200 OK** response. b. body returned must accomplish **ServiceSecurity** data structure.

Test Case 6: retrieve the Security Context of an API Invoker with invalid apiInvokerId

Test ID	capif_api_security-6
Description	This test case will check that a provider can retrieve the Security context of an API Invoker
Pre-conditions	The Administrator must have previously registered the User The user must have the access token, the ca root certificate and the URLs to onboard an Invoker. API Invoker has created a valid Security Context
Execution Steps	1. Onboard Invoker at CCF. 2. Onboard Provider at CCF. 3. Create Security Context using invoker Certificate 4. Retrieve Security Context by Provider of invalid invoker

Information of Test	1. Onboard Invoker and provider at CCF. 2. Create Security Context for this Invoker: a. Send **PUT** https://{CAPIF_HOSTNAME}/trustedInvokers/{apiInvokerId} b. Use Invoker **Certificate** 3. Retrieve Security Context of Invoker by Provider: a. Send **GET** https://{CAPIF_HOSTNAME}/trustedInvokers/{API_INVOKER_NOT_VALID} b. Using **AEF Certificate**
Expected Result	1. Retrieve security context: a. **404 Not Found** response. b. body returned must accomplish **ProblemDetails** data structure, with: • status **404** • title with message "Not Found" • detail with message "Invoker not found". • cause with message "API Invoker not exists or invalid ID".

Test Case 7: Retrieve the Security Context of an API Invoker with invalid apfId

Test ID	capif_api_security-7
Description	This test case will check that a provider cannot retrieve the Security context of an API Invoker without valid apfId
Pre-conditions	The Administrator must have previously registered the User The user must have the access token, the ca root certificate and the URLs to onboard an Invoker. API Invoker has created a valid Security Context
Execution Steps	1. Onboard Invoker at CCF. 2. Onboard Provider at CCF. 3. Create Security Context using invoker Certificate 4. Retrieve Security Context by Provider
Information of Test	1. Onboard Invoker and provider at CCF. 2. Create Security Context for this Invoker: a. Send **PUT** https://{CAPIF_HOSTNAME}/trustedInvokers/{apiInvokerId} b. Use Invoker **Certificate** 3. Retrieve Security Context of Invoker by Provider: a. Send **GET** https://{CAPIF_HOSTNAME}/trustedInvokers/{apiInvokerId} b. Using **Invoker Certificate**
Expected Result	1. Retrieve security context: a. **401 Unauthorized** response. b. body returned must accomplish **ProblemDetails** data structure, with: • status **401** • title with message "Unauthorized" • detail with message "Role not authorized for this API route". • cause with message "User role must be aef".

Test Case 8: Delete the Security Context of an API Invoker

Test ID	capif_api_security-8
Description	This test case will check that a provider can delete a Security context
Pre-conditions	The Administrator must have previously registered the User The user must have the access token, the ca root certificate and the URLs to onboard an Invoker. API Invoker has created a valid Security Context

Execution Steps	1. Onboard Invoker at CCF. 2. Onboard Provider at CCF. 3. Create Security Context using invoker Certificate 4. Delete Security Context by Provider

Information of Test	1. Onboard Invoker and provider at CCF.
	2. Create Security Context for this Invoker:
	  a. Send **PUT** https://{CAPIF_HOSTNAME}/trustedInvokers/{apiInvokerId}
	  b. Use Invoker **Certificate**
	3. Delete Security Context of Invoker by Provider:
	  a. Send **DELETE** https://{CAPIF_HOSTNAME}/trustedInvokers/{apiInvokerId}
	  b. Use **AEF Certificate**
	4. Retrieve Security Context of Invoker by Provider:
	  a. Send **GET** https://{CAPIF_HOSTNAME}/trustedInvokers/{apiInvokerId}
	  b. Using **AEF Certificate**

Expected Result	1. Delete Security Context: a. **204 No Content** response 2. Retrieve security context: a. **404 Not Found** response. b. body returned must accomplish **ProblemDetails** data structure, with: • status **404** • title with message "Not Found" • detail with message "Security context not found". • cause with message "API Invoker not exists or invalid ID".

Test Case 9: Delete the Security Context of an API Invoker with Invoker entity role

Test ID	capif_api_security-9
Description	This test case will check that an Invoker cannot delete a Security context
Pre-conditions	The Administrator must have previously registered the User The user must have the access token, the ca root certificate and the URLs to onboard an Invoker. API Invoker has created a valid Security Context
Execution Steps	1. Onboard Invoker at CCF. 2. Onboard Provider at CCF. 3. Create Security Context using invoker Certificate 4. Delete Security Context by Invoker

Information of Test	1. Onboard Invoker and provider at CCF.
	2. Create Security Context for this Invoker:
	  a. Send **PUT** https://{CAPIF_HOSTNAME}/trustedInvokers/{apiInvokerId}
	  b. Use Invoker **Certificate**
	3. Delete Security Context of Invoker by invoker:
	  a. Send **DELETE** https://{CAPIF_HOSTNAME}/trustedInvokers/{apiInvokerId}
	  b. Use **Invoker Certificate**

Expected Result	1. Delete Security Contex by invokert: a. **401 Unauthorized** response. b. body returned must accomplish **ProblemDetails** data structure, with: • status **401** • title with message "Unauthorized" • detail with message "Role not authorized for this API route". • cause with message "User role must be aef".

Test Case 10: Delete the Security Context of an API Invoker with Invoker entity role and invalid apiInvokerId

Test ID	capif_api_security-10
Description	This test case will check that an Invoker cannot delete a Security context with invalid apiInvokerId.
Pre-conditions	The Administrator must have previously registered the User The user must have the access token, the ca root certificate and the URLs to onboard an Invoker. API Invoker has created a valid Security Context
Execution Steps	1. Onboard Invoker at CCF. 2. Onboard Provider at CCF. 3. Create Security Context using invoker Certificate 4. Delete Security Context by Invoker with invalid apiInvokerId
Information of Test	4. Onboard Invoker and provider at CCF. 5. Create Security Context for this Invoker: a. Send **PUT** https://{CAPIF_HOSTNAME}/trustedInvokers/{apiInvokerId} b. Use Invoker **Certificate** 6. Delete Security Context of Invoker by invoker: a. Send **DELETE** https://{CAPIF_HOSTNAME}/trustedInvokers/{API_INVOKER_NOT_VALID} b. Use **Invoker Certificate**
Expected Result	2. Delete Security Contex by invokert: a. **401 Unauthorized** response. b. body returned must accomplish **ProblemDetails** data structure, with: • status **401** • title with message "Unauthorized" • detail with message "Role not authorized for this API route". • cause with message "User role must be aef".

Test Case 11: Delete the Security Context of an API Invoker with invalid apiInvokerId

Test ID	capif_api_security-11
Description	This test case will check that an Provider cannot delete a Security context of invalid apiInvokerId
Pre-conditions	The Administrator must have previously registered the User The user must have the access token, the ca root certificate and the URLs to onboard an Invoker. API Invoker has created a valid Security Context
Execution Steps	1. Onboard Invoker at CCF. 2. Onboard Provider at CCF. 3. Create Security Context using invoker Certificate 4. Delete Security Context by Provider with invalid apiInvokerId
Information of Test	1. Onboard Invoker and provider at CCF. 2. Create Security Context for this Invoker: a. Send **PUT** https://{CAPIF_HOSTNAME}/trustedInvokers/{apiInvokerId} b. Use Invoker **Certificate** 3. Delete Security Context of Invoker by invoker: a. Send **DELETE** https://{CAPIF_HOSTNAME}/trustedInvokers/{NOT_VALID_API_INVOKER_ID} b. Use AEF **Certificate**

Expected Result	1. Delete Security Contex by invokert: a. **404 Not Found** response. b. body returned must accomplish **ProblemDetails** data structure, with: • status **404** • title with message "Not Found" • detail with message "Invoker not found". • cause with message "API Invoker not exists or invalid ID".

Test Case 12: Update the Security Context of an API Invoker

Test ID	capif_api_security-12
Description	This test case will check that an API Invoker can update a Security context
Pre-conditions	The Administrator must have previously registered the User The user must have the access token, the ca root certificate and the URLs to onboard an Invoker. API Invoker has created a valid Security Context
Execution Steps	1. Onboard Invoker at CCF. 2. Onboard Provider at CCF. 3. Create Security Context using invoker Certificate 4. Update Security context by invoker 5. Retrieve Security Context By Provider
Information of Test	1. Onboard Invoker and provider at CCF. 2. Create Security Context for this Invoker: a. Send **PUT** https://{CAPIF_HOSTNAME}/trustedInvokers/{apiInvokerId} b. Use Invoker **Certificate** 3. Update Security Context of Invoker: a. Send **POST** https://{CAPIF_HOSTNAME}/trustedInvokers/{apiInvokerId}/update b. body service security body but with notification destination modified to http://robot.testing2 c. Using **Invoker Certificate**. 4. Retrieve Security Context of Invoker by Provider: a. Send **GET** https://{CAPIF_HOSTNAME}/trustedInvokers/{apiInvokerId} b. Using **AEF Certificate**.
Expected Result	1. Update Security Context by invoker: a. **200 OK** response. b. body returned must accomplish **ServiceSecurity** data structure. 2. Retrieve security context: a. **200 OK** response. b. body returned must accomplish **ServiceSecurity** data structure. • Check is this returned object match with modified one.

Test Case 13: Update the Security Context of an API Invoker with provider entity role

Test ID	capif_api_security-13
Description	This test case will check that a provider cannot update a Security context
Pre-conditions	The Administrator must have previously registered the User The user must have the access token, the ca root certificate and the URLs to onboard an Invoker. API Invoker has created a valid Security Context
Execution Steps	1. Onboard Invoker at CCF. 2. Onboard Provider at CCF. 3. Create Security Context using invoker Certificate

	4. Update Security context as provider
Information of Test	1. Onboard Invoker and provider at CCF. 2. Create Security Context for this Invoker: a. Send **PUT** https://{CAPIF_HOSTNAME}/trustedInvokers/{apiInvokerId} b. Use Invoker **Certificate** 3. Update Security Context of Invoker: a. Send **POST** https://{CAPIF_HOSTNAME}/trustedInvokers/{apiInvokerId}/update b. body service security body but with notification destination modified to http://robot.testing2 c. Using **AEF Certificate**.
Expected Result	1. Update Security Context by provider: a. **401 Unauthorized** response. b. body returned must accomplish **ProblemDetails** data structure, with: • status **401** • title with message "Unauthorized" • detail with message "Role not authorized for this API route". • cause with message "User role must be aef".

Test Case 14: Update the Security Context of an API Invoker with provider entity role and invalid apiInvokerId

Test ID	capif_api_security-14
Description	This test case will check that a provider cannot update a Security context of invalid apiInvokerId
Pre-conditions	The Administrator must have previously registered the User The user must have the access token, the ca root certificate and the URLs to onboard an Invoker. API Invoker has created a valid Security Context
Execution Steps	5. Onboard Invoker at CCF. 6. Onboard Provider at CCF. 7. Create Security Context using invoker Certificate 8. Update Security context as provider
Information of Test	1. Onboard Invoker and provider at CCF. 2. Create Security Context for this Invoker: a. Send **PUT** https://{CAPIF_HOSTNAME}/trustedInvokers/{apiInvokerId} b. Use Invoker **Certificate** 3. Update Security Context of Invoker: a. Send **POST** https://{CAPIF_HOSTNAME}/trustedInvokers/{apiInvokerId}/update b. body service security body but with notification destination modified to http://robot.testing2 c. Using **AEF Certificate**.
Expected Result	1. Update Security Context by provider: a. **401 Unauthorized** response. b. body returned must accomplish **ProblemDetails** data structure, with: • status **401** • title with message "Unauthorized" • detail with message "Role not authorized for this API route". • cause with message "User role must be aef".

Test Case 15: Update the Security Context of an API Invoker with invalid apiInvokerId

Test ID	capif_api_security-15
Description	This test case will check that an API Invoker cannot update a Security context not valid apiInvokerId

Pre-conditions	The Administrator must have previously registered the User The user must have the access token, the ca root certificate and the URLs to onboard an Invoker. API Invoker has created a valid Security Context
Execution Steps	1. Onboard Invoker at CCF. 2. Onboard Provider at CCF. 3. Create Security Context using invoker Certificate 4. Update Security context as invoker
Information of Test	1. Onboard Invoker and provider at CCF. 2. Create Security Context for this Invoker: a. Send **PUT** https://{CAPIF_HOSTNAME}/trustedInvokers/{API_INVOKER_NOT_VALID} b. Use Invoker **Certificate** 3. Update Security Context of Invoker: a. Send **POST** https://{CAPIF_HOSTNAME}/trustedInvokers/{apiInvokerId}/update b. body service security body but with notification destination modified to http://robot.testing2 c. Using **Invoker Certificate**.
Expected Result	1. Update Security Context: a. **404 Not Found** response. b. body returned must accomplish **ProblemDetails** data structure, with: • status **404** • title with message "Not Found" • detail with message "Invoker not found". • cause with message "API Invoker not exists or invalid ID".

Test Case 16: Revoke the authorization of the API Invoker for APIs

Test ID	capif_api_security-16
Description	This test case will check that a Provider can revoke the authorization for APIs
Pre-conditions	The Administrator must have previously registered the User The user must have the access token, the ca root certificate and the URLs to onboard an Invoker. API Invoker has created a valid Security Context
Execution Steps	1. Onboard Invoker at CCF. 2. Onboard Provider at CCF. 3. Create Security Context using invoker Certificate 4. Revoke Security Context by Provider 5. Retrieve Security Context by Provider
Information of Test	1. Onboard Invoker and provider at CCF. 2. Create Security Context for this Invoker: a. Send **PUT** https://{CAPIF_HOSTNAME}/trustedInvokers/{API_INVOKER_NOT_VALID} b. Use Invoker **Certificate** 3. Revoke Authorization by Provider: a. Send **POST** https://{CAPIF_HOSTNAME}/trustedInvokers/{apiInvokerId}/delete b. Using **AEF Certificate**. 4. Retrieve Security Context by Provider: a. Send **GET** https://{CAPIF_HOSTNAME}/trustedInvokers/{apiInvokerId} b. Using **AEF Certificate**.
Expected Result	1. Revoke authorization: a. **204 No Content** response 2. Retrieve Security Context: a. **404 Not Found** response. b. body returned must accomplish **ProblemDetails** data structure, with: • status **404** • title with message "Not Found"

	• detail with message "Security context not found". • cause with message "API Invoker has not security context".

Test Case 17: Revoke the authorization of the API Invoker for APIs without valid apfId

Test ID	capif_api_security-17
Description	This test case will check that an Invoker can't revoke the authorization for APIs
Pre-conditions	The Administrator must have previously registered the User The user must have the access token, the ca root certificate and the URLs to onboard an Invoker. API Invoker has created a valid Security Context
Execution Steps	1. Onboard Invoker at CCF. 2. Onboard Provider at CCF. 3. Create Security Context using invoker Certificate 4. Revoke Security Context by Invoker 5. Retrieve Security Context by Provider

Information of Test	1. Onboard Invoker and provider at CCF.
	2. Create Security Context for this Invoker:
	  a. Send **PUT** https://{CAPIF_HOSTNAME}/trustedInvokers/{API_INVOKER_NOT_VALID}
	  b. Use Invoker **Certificate**
	3. Revoke Authorization by invoker:
	  a. Send **POST** https://{CAPIF_HOSTNAME}/trustedInvokers/{apiInvokerId}/delete
	  b. Using **Invoker Certificate**.
	4. Retrieve Security Context by Provider:
	  a. Send **GET** https://{CAPIF_HOSTNAME}/trustedInvokers/{apiInvokerId}
	  b. Using **AEF Certificate**.

Expected Result	1. Revoke authorization: a. **401 Unauthorized** response. b. body returned must accomplish **ProblemDetails** data structure, with: • status **401** • title with message "Unauthorized" • detail with message "Role not authorized for this API route". • cause with message "User role must be provider". 2. Retrieve Security Context: a. **200 OK** response. b. body returned must accomplish **ServiceSecurity** data structure. • Check is this returned object match with created one.

Test Case 18: Revoke the authorization of the API Invoker for APIs with invalid apiInvokerId

Test ID	capif_api_security-18
Description	This test case will check that an API Exposure Function cannot revoke the authorization for APIs for invalid apiInvokerId
Pre-conditions	The Administrator must have previously registered the User The user must have the access token, the ca root certificate and the URLs to onboard an Invoker. API Invoker has created a valid Security Context
Execution Steps	1. Onboard Invoker at CCF. 2. Onboard Provider at CCF. 3. Create Security Context using invoker Certificate

	4. Revoke Security Context by Provider wit invalid apiInvokerId 5. Retrieve Security Context by Provider

Information of Test	1. Onboard Invoker and provider at CCF.
	2. Create Security Context for this Invoker:
	  a. Send **PUT** https://{CAPIF_HOSTNAME}/trustedInvokers/{API_INVOKER_NOT_VALID}
	  b. Use Invoker **Certificate**
	3. Revoke Authorization by provider:
	  a. Send **POST** https://{CAPIF_HOSTNAME}/trustedInvokers/{API_INVOKER_NOT_VALID}/delete
	  b. Using **Invoker Certificate**.
	4. Retrieve Security Context by Provider:
	  a. Send **GET** https://{CAPIF_HOSTNAME}/trustedInvokers/{apiInvokerId}
	  b. Using **AEF Certificate**.

Expected Result	1. Revoke authorization: a. **404 Not Found** response. b. body returned must accomplish **ProblemDetails** data structure, with: • status **404** • title with message "Not Found" • detail with message "Invoker not found". • cause with message "API Invoker not exists or invalid ID". 2. Retrieve Security Context: a. **200 OK** response. b. body returned must accomplish **ServiceSecurity** data structure. • Check is this returned object match with created one.

Test Case 19: Retrieve access token

Test ID	capif_api_security-19
Description	This test case will check that an API Invoker can retrieve a security access token OAuth 2.0.
Pre-conditions	The Administrator must have previously registered the User The user must have the access token, the ca root certificate and the URLs to onboard an Invoker. API Invoker has created a valid Security Context
Execution Steps	1. Onboard Provider and publish Service API service_1 at CCF. 2. Onboard Invoker at CCF. 3. Discover service APIs by Invoker 4. Create Security Context using invoker Certificate 5. Request Access token

Information of Test	1. Onboard Invoker and provider at CCF.
	2. Publish Service API at CCF:
	  a. Send **POST** to ccf_publish_url https://{CAPIF_HOSTNAME}/published-apis/v1/{apfId}/service-apis
	  b. body with apiName **service_1**
	  c. Use **APF Certificate**
	3. Request Discover Published APIs not filtered:
	  a. Send **GET** to ccf_discover_url https://{CAPIF_HOSTNAME}/service-apis/v1/allServiceAPIs?api-invoker-id={apiInvokerId}
	  b. Param api-invoker-id is mandatory
	  c. Using **Invoker Certificate**
	4. Create Security Context for this Invoker
	  a. Send **PUT** https://{CAPIF_HOSTNAME}/trustedInvokers/{apiInvokerId}
	  b. Using **Invoker Certificate**.
	  c. Create Security Information Body with one **securityInfo** for each aef present at each serviceAPIDescription present at Discover.
	5. Request Access Token by invoker:
	  a. Sent POST https://{CAPIF_HOSTNAME}/securities/{securityId}/token:

	b. body access token req body and example example c. **securityId** is apiInvokerId. d. **grant_type=client_credentials**. e. Create Scope properly for request: **3gpp#{aef_id}:{api_name}** f. Using **Invoker Certificate**.
Expected Result	1. Response to request access token: a. **200 OK** b. body must follow **AccessTokenRsp** with: • access_token present • token_type=Bearer

Test Case 20: Retrieve access token by provider

Test ID	capif_api_security-20
Description	This test case will check that an API Provider cannot retrieve a security access token OAuth 2.0.
Pre-conditions	The Administrator must have previously registered the User The user must have the access token, the ca root certificate and the URLs to onboard an Invoker. API Invoker has created a valid Security Context
Execution Steps	1. Onboard Provider and publish Service API service_1 at CCF. 2. Onboard Invoker at CCF. 3. Discover service APIs by Invoker 4. Create Security Context using invoker Certificate 5. Request Access token as Provider
Information of Test	1. Onboard Invoker and provider at CCF. 2. Publish Service API at CCF: a. Send **POST** to ccf_publish_url https://{CAPIF_HOSTNAME}/published-apis/v1/{apfId}/service-apis b. body with apiName **service_1** c. Use **APF Certificate** 3. Request Discover Published APIs not filtered: a. Send **GET** to ccf_discover_url https://{CAPIF_HOSTNAME}/service-apis/v1/allServiceAPIs?api-invoker-id={apiInvokerId} b. Param api-invoker-id is mandatory c. Using **Invoker Certificate** 4. Create Security Context for this Invoker a. Send **PUT** https://{CAPIF_HOSTNAME}/trustedInvokers/{apiInvokerId} b. Using **Invoker Certificate**. c. Create Security Information Body with one **securityInfo** for each aef present at each serviceAPIDescription present at Discover. 5. Request Access Token by invoker: a. Sent POST https://{CAPIF_HOSTNAME}/securities/{securityId}/token: b. body access token req body c. **securityId** is apiInvokerId. d. **grant_type=client_credentials**. e. Create Scope properly for request: **3gpp#{aef_id}:{api_name}** f. Using **AEF Certificate**.
Expected Result	1. Response to request access token: a. **401 Unauthorized** response. b. body returned must accomplish **AccessTokenErr** data structure, with: • error unauthorized_client • error_description=Role not authorized for this API route

Test Case 21: Retrieve access token by provider with invalid apiInvokerId

Test ID	capif_api_security-21
Description	This test case will check that an API Exposure Function cannot retrieve a security access token without valid apiInvokerId
Pre-conditions	The Administrator must have previously registered the User The user must have the access token, the ca root certificate and the URLs to onboard an Invoker. API Invoker has created a valid Security Context
Execution Steps	1. Onboard Provider and publish Service API service_1 at CCF. 2. Onboard Invoker at CCF. 3. Discover service APIs by Invoker 4. Create Security Context using invoker Certificate 5. Request Access token as Provider with invalid apiInvokerId
Information of Test	1. Onboard Invoker and provider at CCF. 2. Publish Service API at CCF: a. Send **POST** to ccf_publish_url https://{CAPIF_HOSTNAME}/published-apis/v1/{apfId}/service-apis b. body with apiName **service_1** c. Use **APF Certificate** 3. Request Discover Published APIs not filtered: a. Send **GET** to ccf_discover_url https://{CAPIF_HOSTNAME}/service-apis/v1/allServiceAPIs?api-invoker-id={apiInvokerId} b. Param api-invoker-id is mandatory c. Using **Invoker Certificate** 4. Create Security Context for this Invoker a. Send **PUT** https://{CAPIF_HOSTNAME}/trustedInvokers/{apiInvokerId} b. Using **Invoker Certificate**. c. Create Security Information Body with one **securityInfo** for each aef present at each serviceAPIDescription present at Discover. 5. Request Access Token by invoker: a. Sent POST https://{CAPIF_HOSTNAME}/securities/{API_INVOKER_ID_NOT_VALID}/token: b. body access token req body c. **securityId** is apiInvokerId. d. **grant_type=client_credentials**. e. Create Scope properly for request: **3gpp#{aef_id}:{api_name}** f. Using **AEF Certificate**.
Expected Result	1. Response to request access token: a. **401 Unauthorized** response. b. body returned must accomplish **AccessTokenErr** data structure, with: • error unauthorized_client • error_description=Role not authorized for this API route

Test Case 22: Retrieve access token with invalid apiInvokerId

Test ID	capif_api_security-22
Description	This test case will check that an API Invoker can't retrieve a security access token without valid apiInvokerId
Pre-conditions	The Administrator must have previously registered the User The user must have the access token, the ca root certificate and the URLs to onboard an Invoker. API Invoker has created a valid Security Context

Execution Steps	1. Onboard Provider and publish Service API service_1 at CCF. 2. Onboard Invoker at CCF. 3. Discover service APIs by Invoker 4. Create Security Context using invoker Certificate 5. Request Access token as invoker with invalid apiInvokerId
Information of Test	1. Onboard Invoker and provider at CCF. 2. Publish Service API at CCF: a. Send **POST** to ccf_publish_url https://{CAPIF_HOSTNAME}/published-apis/v1/{apfId}/service-apis b. body with apiName **service_1** c. Use **APF Certificate** 3. Request Discover Published APIs not filtered: a. Send **GET** to ccf_discover_url https://{CAPIF_HOSTNAME}/service-apis/v1/allServiceAPIs?api-invoker-id={apiInvokerId} b. Param api-invoker-id is mandatory c. Using **Invoker Certificate** 4. Create Security Context for this Invoker a. Send **PUT** https://{CAPIF_HOSTNAME}/trustedInvokers/{apiInvokerId} b. Using **Invoker Certificate**. c. Create Security Information Body with one **securityInfo** for each aef present at each serviceAPIDescription present at Discover. 5. Request Access Token by invoker: a. Sent POST https://{CAPIF_HOSTNAME}/securities/{API_INVOKER_ID_NOT_VALID}/token: b. body access token req body c. **securityId** is apiInvokerId. d. **grant_type=client_credentials**. e. Create Scope properly for request: **3gpp#{aef_id}:{api_name}** f. Using **Invoker Certificate**.
Expected Result	1. Response to request access token: a. **404 Not Found** response. b. body returned must accomplish **ProblemDetails29571** data structure, with: • status 404 • title Not Found • detail Security context not found • cause API Invoker has no security context

Test Case 23: Retrieve access token with invalid client_id

Test ID	capif_api_security-23
Description	This test case will check that an API Exposure Function cannot retrieve a security access token without valid client_id at body
Pre-conditions	The Administrator must have previously registered the User The user must have the access token, the ca root certificate and the URLs to onboard an Invoker. API Invoker has created a valid Security Context
Execution Steps	1. Onboard Provider and publish Service API service_1 at CCF. 2. Onboard Invoker at CCF. 3. Discover service APIs by Invoker 4. Create Security Context using invoker Certificate 5. Request Access token as invoker
Information of Test	1. Onboard Invoker and provider at CCF. 2. Publish Service API at CCF: a. Send **POST** to ccf_publish_url https://{CAPIF_HOSTNAME}/published-apis/v1/{apfId}/service-apis b. body with apiName **service_1**

	c. Use **APF Certificate** 3. Request Discover Published APIs not filtered: a. Send **GET** to ccf_discover_url https://{CAPIF_HOSTNAME}/service-apis/v1/allServiceAPIs?api-invoker-id={apiInvokerId} b. Param api-invoker-id is mandatory c. Using **Invoker Certificate** 4. Create Security Context for this Invoker a. Send **PUT** https://{CAPIF_HOSTNAME}/trustedInvokers/{apiInvokerId} b. Using **Invoker Certificate**. c. Create Security Information Body with one **securityInfo** for each aef present at each serviceAPIDescription present at Discover. 5. Request Access Token by invoker: a. Sent POST https://{CAPIF_HOSTNAME}/securities/{apiInvokerId}/token: b. body access token req body c. **securityId** is apiInvokerId. d. **grant_type=client_credentials**. e. **Client_id** is not valid f. Create Scope properly for request: **3gpp#{aef_id}:{api_name}** g. Using **Invoker Certificate**.
Expected Result	1. Response to request access token: a. **400 Bad Request** response. b. body returned must accomplish **AccessTokenErr** data structure, with: • error invalid_client • error_description=Client Id not found

Test Case 24: Retrieve access token with unsupported grant_type

Test ID	capif_api_security-24
Description	This test case will check that an API Exposure Function cannot retrieve a security access token with unsupported grant_type
Pre-conditions	The Administrator must have previously registered the User The user must have the access token, the ca root certificate and the URLs to onboard an Invoker. API Invoker has created a valid Security Context
Execution Steps	1. Onboard Provider and publish Service API service_1 at CCF. 2. Onboard Invoker at CCF. 3. Discover service APIs by Invoker 4. Create Security Context using invoker Certificate 5. Request Access token as invoker
Information of Test	1. Onboard Invoker and provider at CCF. 2. Publish Service API at CCF: a. Send **POST** to ccf_publish_url https://{CAPIF_HOSTNAME}/published-apis/v1/{apfId}/service-apis b. body with apiName **service_1** c. Use **APF Certificate** 3. Request Discover Published APIs not filtered: a. Send **GET** to ccf_discover_url https://{CAPIF_HOSTNAME}/service-apis/v1/allServiceAPIs?api-invoker-id={apiInvokerId} b. Param api-invoker-id is mandatory c. Using **Invoker Certificate** 4. Create Security Context for this Invoker a. Send **PUT** https://{CAPIF_HOSTNAME}/trustedInvokers/{apiInvokerId} b. Using **Invoker Certificate**. c. Create Security Information Body with one **securityInfo** for each aef present at each serviceAPIDescription present at Discover. 5. Request Access Token by invoker: a. Sent POST https://{CAPIF_HOSTNAME}/securities/{apiInvokerId}/token:

	b. body access token req body c. **securityId** is apiInvokerId. d. **grant_type=not_valid**. e. Create Scope properly for request: **3gpp#{aef_id}:{api_name}** f. Using **Invoker Certificate**.
Expected Result	1. Response to request access token: a. **400 Bad Request** response. b. body returned must accomplish **AccessTokenErr** data structure, with: • error unsupported_grant_type • error_description=Invalid value for `grant_type` \(${grant_type}\), must be one of \['client_credentials'\] - 'grant_type'

Test Case 25: Retrieve access token with invalid scope

Test ID	capif_api_security-25
Description	This test case will check that an API Exposure Function cannot retrieve a security access token with complete invalid scope
Pre-conditions	The Administrator must have previously registered the User The user must have the access token, the ca root certificate and the URLs to onboard an Invoker. API Invoker has created a valid Security Context
Execution Steps	1. Onboard Provider and publish Service API service_1 at CCF. 2. Onboard Invoker at CCF. 3. Discover service APIs by Invoker 4. Create Security Context using invoker Certificate 5. Request Access token as invoker
Information of Test	1. Onboard Invoker and provider at CCF. 2. Publish Service API at CCF: a. Send **POST** to ccf_publish_url https://{CAPIF_HOSTNAME}/published-apis/v1/{apfId}/service-apis b. body with apiName **service_1** c. Use **APF Certificate** 3. Request Discover Published APIs not filtered: a. Send **GET** to ccf_discover_url https://{CAPIF_HOSTNAME}/service-apis/v1/allServiceAPIs?api-invoker-id={apiInvokerId} b. Param api-invoker-id is mandatory c. Using **Invoker Certificate** 4. Create Security Context for this Invoker a. Send **PUT** https://{CAPIF_HOSTNAME}/trustedInvokers/{apiInvokerId} b. Using **Invoker Certificate**. c. Create Security Information Body with one **securityInfo** for each aef present at each serviceAPIDescription present at Discover. 5. Request Access Token by invoker: a. Sent POST https://{CAPIF_HOSTNAME}/securities/{apiInvokerId}/token: b. body access token req body c. **securityId** is apiInvokerId. d. **grant_type=client_credentials**. e. Create Scope properly for request: **not-valid-scope** f. Using **Invoker Certificate**.
Expected Result	1. Response to request access token: a. **400 Bad Request** response. b. body returned must accomplish **AccessTokenErr** data structure, with: • error invalid scope • error_description=The first characters must be '3gpp'

Test Case 26: Retrieve access token with invalid aefId at scope

Test ID	capif_api_security-26
Description	This test case will check that an API Exposure Function cannot retrieve a security access token with invalid aefId at scope
Pre-conditions	The Administrator must have previously registered the User The user must have the access token, the ca root certificate and the URLs to onboard an Invoker. API Invoker has created a valid Security Context
Execution Steps	1. Onboard Provider and publish Service API service_1 at CCF. 2. Onboard Invoker at CCF. 3. Discover service APIs by Invoker 4. Create Security Context using invoker Certificate 5. Request Access token as invoker
Information of Test	1. Onboard Invoker and provider at CCF. 2. Publish Service API at CCF: a. Send **POST** to ccf_publish_url https://{CAPIF_HOSTNAME}/published-apis/v1/{apfId}/service-apis b. body with apiName **service_1** c. Use **APF Certificate** 3. Request Discover Published APIs not filtered: a. Send **GET** to ccf_discover_url https://{CAPIF_HOSTNAME}/service-apis/v1/allServiceAPIs?api-invoker-id={apiInvokerId} b. Param api-invoker-id is mandatory c. Using **Invoker Certificate** 4. Create Security Context for this Invoker a. Send **PUT** https://{CAPIF_HOSTNAME}/trustedInvokers/{apiInvokerId} b. Using **Invoker Certificate**. c. Create Security Information Body with one **securityInfo** for each aef present at each serviceAPIDescription present at Discover. 5. Request Access Token by invoker: a. Sent POST https://{CAPIF_HOSTNAME}/securities/{apiInvokerId}/token: b. body access token req body c. **securityId** is apiInvokerId. d. **grant_type=client_credentials**. e. Create Scope properly for request: **3gpp#1234:*service_1** f. Using **Invoker Certificate**.
Expected Result	1. Response to request access token: a. **400 Bad Request** response. b. body returned must accomplish **AccessTokenErr** data structure, with: • error invalid scope • error_description=One of aef_id does not belong of your security context

Test Case 27: Retrieve access token with invalid apiName at scope

Test ID	capif_api_security-27
Description	This test case will check that an API Exposure Function cannot retrieve a security access token with invalid apiName at scope
Pre-conditions	The Administrator must have previously registered the User The user must have the access token, the ca root certificate and the URLs to onboard an Invoker. API Invoker has created a valid Security Context
Execution Steps	1. Onboard Provider and publish Service API service_1 at CCF. 2. Onboard Invoker at CCF. 3. Discover service APIs by Invoker

	4. Create Security Context using invoker Certificate 5. Request Access token as invoker
Information of Test	1. Onboard Invoker and provider at CCF. 2. Publish Service API at CCF: a. Send **POST** to ccf_publish_url https://{CAPIF_HOSTNAME}/published-apis/v1/{apfId}/service-apis b. body with apiName **service_1** c. Use **APF Certificate** 3. Request Discover Published APIs not filtered: a. Send **GET** to ccf_discover_url https://{CAPIF_HOSTNAME}/service-apis/v1/allServiceAPIs?api-invoker-id={apiInvokerId} b. Param api-invoker-id is mandatory c. Using **Invoker Certificate** 4. Create Security Context for this Invoker a. Send **PUT** https://{CAPIF_HOSTNAME}/trustedInvokers/{apiInvokerId} b. Using **Invoker Certificate**. c. Create Security Information Body with one **securityInfo** for each aef present at each serviceAPIDescription present at Discover. 5. Request Access Token by invoker: a. Sent POST https://{CAPIF_HOSTNAME}/securities/{apiInvokerId}/token: b. body access token req body c. **securityId** is apiInvokerId. d. **grant_type=client_credentials**. e. Create Scope properly for request: **3gpp#{aef_id}:not-valid** f. Using **Invoker Certificate**.
Expected Result	1. Response to request access token: a. **400 Bad Request** response. b. body returned must accomplish **AccessTokenErr** data structure, with: • error invalid scope • error_description=One of the api names does not exist or is not associated with the aef id provided

D.7 Test Plan for CAPIF API Access Control Policy Service

This section list test cases for the CAPIF_Access_Control_Policy_API.

Test Case 1: Retrieve ACL

Test ID	capif_api_acl -1
Description	This test case will check that an API Provider can retrieve ACL from CAPIF
Pre-conditions	The Administrator must have previously registered the User The user must have the access token, the ca root certificate and the URLs to onboard an Invoker. Service published in CAPIF Security context created for Service API published
Execution Steps	1. Onboard Invoker and Provider at CCF. 2. Publish Service API with name service_1 3. Create Security Context 4. Provider Get ACL information
Information of Test	1. Onboard Invoker and provider at CCF. 2. Publish Service API at CCF: a. Send **POST** to ccf_publish_url https://{CAPIF_HOSTNAME}/published-apis/v1/{apfId}/service-apis b. body [service api description] with apiName **service_1** c. Use **APF Certificate**

	3. Create Security Context for this Invoker a. Send **PUT** https://{CAPIF_HOSTNAME}/trustedInvokers/{apiInvokerId} b. Use **Invoker Certificate** 4. Provider Retrieve ACL a. Send **GET** https://{CAPIF_HOSTNAME}/access-control-policy/v1/accessControlPolicyList/${serviceApiId}?aef-id=${aef_id} b. Use **serviceApiId** and **aefId** c. Use **AEF Certificate**
Expected Result	1. Response to ACL Service must accomplish: a. **200 OK** Response. b. body returned must accomplish **AccessControlPolicyList** data structure. c. **apiInvokerPolicies** must: • contain only one object. • apiInvokerId must match apiInvokerId registered previously.

Test Case 2: Retrieve ACL with 2 Service APIs published

Test ID	capif_api_acl-2
Description	This test case will check that an API Provider can retrieve ACL from CAPIF for 2 different serviceApis published.
Pre-conditions	The Administrator must have previously registered the User The user must have the access token, the ca root certificate and the URLs to onboard an Invoker. Services API published in CAPIF Security context created for Service API published
Execution Steps	1. Onboard Invoker and Provider at CCF. 2. Publish Service API with name service_1 and service_2 3. Create Security Context 4. Provider Get ACL information for service_1 5. Provider Get ACL information for service_2
Information of Test	1. Onboard Invoker and provider at CCF. 2. Publish Service API at CCF: a. Send **POST** to ccf_publish_url https://{CAPIF_HOSTNAME}/published-apis/v1/{apfId}/service-apis b. body [service api description] with apiName **service_1** c. Use **APF Certificate** 3. Publish Service API at CCF: a. Send **POST** to ccf_publish_url https://{CAPIF_HOSTNAME}/published-apis/v1/{apfId}/service-apis b. body [service api description] with apiName **service_2** c. Use **APF Certificate** 4. Create Security Context for both published APIs a. Send **PUT** https://{CAPIF_HOSTNAME}/trustedInvokers/{apiInvokerId} b. Use **Invoker Certificate** 5. Provider Retrieve ACL for serviceApiId1: a. Send **GET** https://{CAPIF_HOSTNAME}/access-control-policy/v1/accessControlPolicyList/${serviceApiId1}?aef-id=${aef_id} b. Use **serviceApiId** and **aefId** c. Use **AEF Certificate** 6. Provider Retrieve ACL for serviceApiId2: a. Send **GET** https://{CAPIF_HOSTNAME}/access-control-policy/v1/accessControlPolicyList/${serviceApiId2}?aef-id=${aef_id} b. Use **serviceApiId** and **aefId** c. Use **AEF Certificate**
Expected Result	1. Response to ACL Service for service_1 must accomplish: a. **200 OK** Response. b. body returned must accomplish **AccessControlPolicyList** data structure.

	c. **apiInvokerPolicies** must: • contain only one object. • apiInvokerId must match apiInvokerId1 registered previously. 2. Response to ACL Service for service_2 must accomplish: a. **200 OK** Response. b. body returned must accomplish **AccessControlPolicyList** data structure. c. **apiInvokerPolicies** must: • contain only one object. • apiInvokerId must match apiInvokerId2 registered previously.

Test Case 3: Retrieve ACL with security context created by two different Invokers

Test ID	capif_api_acl-3
Description	This test case will check that an API Provider can retrieve ACL from CAPIF containing 2 objects.
Pre-conditions	The Administrator must have previously registered the User The user must have the access token, the ca root certificate and the URLs to onboard an Invoker. Service API published in CAPIF Security context created by two API Invokers for Service API published
Execution Steps	1. Onboard Invoker and Provider at CCF. 2. Publish Service API with name service_1 and service_2 3. Create Security Context 4. Provider Get ACL information for service_1 5. Provider Get ACL information for service_2
Information of Test	1. Onboard two Invoker and provider at CCF. 2. Publish Service API at CCF: a. Send **POST** to ccf_publish_url https://{CAPIF_HOSTNAME}/published-apis/v1/{apfId}/service-apis b. body [service api description] with apiName **service_1** c. Use **APF Certificate** 3. Create Security Context for each invoker: a. Send **PUT** https://{CAPIF_HOSTNAME}/trustedInvokers/{apiInvokerId} b. Use **Invoker Certificate** 4. Provider Retrieve ACL for serviceApiId: a. Send **GET** https://{CAPIF_HOSTNAME}/access-control-policy/v1/accessControlPolicyList/${serviceApiId1}?aef-id=${aef_id} b. Use **serviceApiId** and **aefId** c. Use **AEF Certificate**
Expected Result	1. Response to ACL Service must accomplish: a. **200 OK** Response. b. body returned must accomplish **AccessControlPolicyList** data structure. c. **apiInvokerPolicies** must: • contain two objects. • One object must match with apiInvokerId1 and the other one with apiInvokerId2.

Test Case 4: Retrieve ACL filtered by api-invoker-id

Test ID	capif_api_acl-4
Description	This test case will check that an API Provider can retrieve ACL filtering by apiInvokerId from CAPIF containing 1 objects.

Pre-conditions	The Administrator must have previously registered the User The user must have the access token, the ca root certificate and the URLs to onboard an Invoker. Service API published in CAPIF Security context created by two API Invokers for Service API published
Execution Steps	1. Onboard Invoker and Provider at CCF. 2. Publish Service API with name service_1 and service_2 3. Create Security Context 4. Provider Get ACL information for service_1 indicating first api-invoker-id 5. Provider Get ACL information for service_2 indicating second api-invoker-id
Information of Test	1. Onboard two Invoker and provider at CCF. 2. Publish Service API at CCF: a. Send **POST** to ccf_publish_url https://{CAPIF_HOSTNAME}/published-apis/v1/{apfId}/service-apis b. body [service api description] with apiName **service_1** c. Use **APF Certificate** 3. Publish another Service API at CCF: a. Send **POST** to ccf_publish_url https://{CAPIF_HOSTNAME}/published-apis/v1/{apfId}/service-apis b. body [service api description] with apiName **service_2** c. Use **APF Certificate** 4. Create Security Context for each invoker for both published APIs: a. Send **PUT** https://{CAPIF_HOSTNAME}/trustedInvokers/{apiInvokerId} b. Use **Invoker Certificate** 5. Provider Retrieve ACL for serviceApiId1: a. Send **GET** https://{CAPIF_HOSTNAME}/access-control-policy/v1/accessControlPolicyList/${serviceApiId1}?aef-id=${aef_id}**&api-invoker-id={apiInvokerId1}** b. Use **serviceApiId, aefId and apiInvokerId1** c. Use **AEF Certificate** 6. Provider Retrieve ACL for serviceApiId2: a. Send **GET** https://{CAPIF_HOSTNAME}/access-control-policy/v1/accessControlPolicyList/${serviceApiId1}?aef-id=${aef_id}**&api-invoker-id={apiInvokerId2}** b. Use **serviceApiId, aefId and apiInvokerId2** c. Use **AEF Certificate**
Expected Result	1. Response to ACL Service must accomplish: a. **200 OK** Response. b. body returned must accomplish **AccessControlPolicyList** data structure. c. **apiInvokerPolicies** must: • contain two objects. • One object must match with apiInvokerId1 2. Response to ACL Service must accomplish: a. **200 OK** Response. b. body returned must accomplish **AccessControlPolicyList** data structure. c. **apiInvokerPolicies** must: • contain two objects. • One object must match with apiInvokerId2

Test Case 5: Retrieve ACL filtered by supported-features

Test ID	capif_api_acl-5
Description	This test case will check that an API Provider can retrieve ACL filtering by supportedFeatures from CAPIF containing 1 objects.
Pre-conditions	The Administrator must have previously registered the User The user must have the access token, the ca root certificate and the URLs to onboard an Invoker.

	Service API published in CAPIF Security context created by two API Invokers for Service API published
Execution Steps	1. Onboard Invoker and Provider at CCF. 2. Publish Service API with name service_1 3. Create Security Context 4. Provider Get ACL information indicating first supported-features 5. Provider Get ACL information indicating second supported-features
Information of Test	1. Onboard two Invoker and provider at CCF. 2. Publish Service API at CCF: a. Send **POST** to ccf_publish_url https://{CAPIF_HOSTNAME}/published-apis/v1/{apfId}/service-apis b. body [service api description] with apiName **service_1** c. Use **APF Certificate** 3. Create Security Context for each invoker for both published APIs: a. Send **PUT** https://{CAPIF_HOSTNAME}/trustedInvokers/{apiInvokerId} b. Use **Invoker Certificate** 4. Provider Retrieve ACL for serviceApiId1: a. Send **GET** https://{CAPIF_HOSTNAME}/access-control-policy/v1/accessControlPolicyList/${serviceApiId1}?aef-id=${aef_id}**&supported-features={apiInvokerId1}** b. Use **serviceApiId, aefId and apiInvokerId1** c. Use **AEF Certificate** 5. Provider Retrieve ACL for serviceApiId2: a. Send **GET** https://{CAPIF_HOSTNAME}/access-control-policy/v1/accessControlPolicyList/${serviceApiId1}?aef-id=${aef_id}**& supported-features={apiInvokerId2}** b. Use **serviceApiId, aefId and apiInvokerId2** c. Use **AEF Certificate**
Expected Result	1. Response to ACL Service must accomplish: a. **200 OK** Response. b. body returned must accomplish **AccessControlPolicyList** data structure. c. **apiInvokerPolicies** must: • contain two objects. • One object must match with supportedFeatures1 2. Response to ACL Service must accomplish: a. **200 OK** Response. b. body returned must accomplish **AccessControlPolicyList** data structure. c. **apiInvokerPolicies** must: • contain two objects. • One object must match with supportedfeatures2

Test Case 6: Retrieve ACL with aef-id not valid

Test ID	capif_api_acl -6
Description	This test case will check that an API Provider can't retrieve ACL from CAPIF if aef-id is not valid
Pre-conditions	The Administrator must have previously registered the User The user must have the access token, the ca root certificate and the URLs to onboard an Invoker. Service published in CAPIF Security context created for Service API published
Execution Steps	1. Onboard Invoker and Provider at CCF. 2. Publish Service API with name service_1 3. Create Security Context 4. Provider Get ACL information

Information of Test	1. Onboard Invoker and provider at CCF. 2. Publish Service API at CCF: a. Send **POST** to ccf_publish_url https://{CAPIF_HOSTNAME}/published-apis/v1/{apfId}/service-apis b. body [service api description] with apiName **service_1** c. Use **APF Certificate** 3. Create Security Context for this Invoker a. Send **PUT** https://{CAPIF_HOSTNAME}/trustedInvokers/{apiInvokerId} b. Use **Invoker Certificate** 4. Provider Retrieve ACL a. Send **GET** https://{CAPIF_HOSTNAME}/access-control-policy/v1/accessControlPolicyList/${serviceApiId}?aef-id=${AEF_ID_NOT_VALID} b. Use **serviceApiId** and **AEF_ID_NOT_VALID** c. Use **AEF Certificate**
Expected Result	1. Response to ACL Service must accomplish: a. **404 Not Found** Response. b. body returned must accomplish **Problem Details** data structure. c. **apiInvokerPolicies** must: • status **404** • title with message "Not Found" • detail with message "No ACLs found for the requested service: {service_api_id}, aef_id: {aef_id}, invoker: {api_invoker_id} and supportedFeatures: {supported_features}". • cause with message "Wrong id".

Test Case 7: Retrieve ACL with service-id not valid

Test ID	capif_api_acl -7
Description	This test case will check that an API Provider can't retrieve ACL from CAPIF if service-api-id is not valid
Pre-conditions	The Administrator must have previously registered the User The user must have the access token, the ca root certificate and the URLs to onboard an Invoker. Service published in CAPIF Security context created for Service API published
Execution Steps	1. Onboard Invoker and Provider at CCF. 2. Publish Service API with name service_1 3. Create Security Context 4. Provider Get ACL information
Information of Test	1. Onboard Invoker and provider at CCF. 2. Publish Service API at CCF: a. Send **POST** to ccf_publish_url https://{CAPIF_HOSTNAME}/published-apis/v1/{apfId}/service-apis b. body [service api description] with apiName **service_1** c. Use **APF Certificate** 3. Create Security Context for this Invoker a. Send **PUT** https://{CAPIF_HOSTNAME}/trustedInvokers/{apiInvokerId} b. Use **Invoker Certificate** 4. Provider Retrieve ACL a. Send **GET** https://{CAPIF_HOSTNAME}/access-control-policy/v1/accessControlPolicyList/${NOT_VALID_SERVICE_API_ID}?aef-id=${aefId} b. Use **NOT_VALID_SERVICE_API_ID** and **aef_id** c. Use **AEF Certificate**

Expected Result	1. Response to ACL Service must accomplish: a. **404 Not Found** Response. b. body returned must accomplish **Problem Details** data structure. c. **apiInvokerPolicies** must: • status **404** • title with message "Not Found" • detail with message "No ACLs found for the requested service: {service_api_id}, aef_id: {aef_id}, invoker: {api_invoker_id} and supportedFeatures: {supported_features}". • cause with message "Wrong id".

Test Case 8: Retrieve ACL with service-id and aef-id not valid

Test ID	capif_api_acl -8
Description	This test case will check that an API Provider can't retrieve ACL from CAPIF if service-api-id and aef-id are not valid
Pre-conditions	The Administrator must have previously registered the User The user must have the access token, the ca root certificate and the URLs to onboard an Invoker. Service published in CAPIF Security context created for Service API published
Execution Steps	1. Onboard Invoker and Provider at CCF. 2. Publish Service API with name service_1 3. Create Security Context 4. Provider Get ACL information
Information of Test	1. Onboard Invoker and provider at CCF. 2. Publish Service API at CCF: a. Send **POST** to ccf_publish_url https://{CAPIF_HOSTNAME}/published-apis/v1/{apfId}/service-apis b. body [service api description] with apiName **service_1** c. Use **APF Certificate** 3. Create Security Context for this Invoker a. Send **PUT** https://{CAPIF_HOSTNAME}/trustedInvokers/{apiInvokerId} b. Use **Invoker Certificate** 4. Provider Retrieve ACL a. Send **GET** https://{CAPIF_HOSTNAME}/access-control-policy/v1/accessControlPolicyList/${NOT_VALID_SERVICE_API_ID}?aef-id=$AEF_ID_NOT_VALID} b. Use **NOT_VALID_SERVICE_API_ID** and **AEF_ID_NOT_VALID** c. Use **AEF Certificate**
Expected Result	1. Response to ACL Service must accomplish: a. **404 Not Found** Response. b. body returned must accomplish **Problem Details** data structure. c. **apiInvokerPolicies** must: • status **404** • title with message "Not Found" • detail with message "No ACLs found for the requested service: {service_api_id}, aef_id: {aef_id}, invoker: {api_invoker_id} and supportedFeatures: {supported_features}". • cause with message "Wrong id".

Test Case 9: Retrieve ACL without Security Context created previously by Invoker

Test ID	capif_api_acl -9
Description	This test case will check that an API Provider can't retrieve ACL if no invoker had requested Security Context to CAPIF
Pre-conditions	The Administrator must have previously registered the User The user must have the access token, the ca root certificate and the URLs to onboard an Invoker. Service published in CAPIF Security context not created for Service API published
Execution Steps	1. Onboard Invoker and Provider at CCF. 2. Publish Service API with name service_1 3. Provider Get ACL information
Information of Test	1. Onboard Invoker and provider at CCF. 2. Publish Service API at CCF: a. Send **POST** to ccf_publish_url https://{CAPIF_HOSTNAME}/published-apis/v1/{apfId}/service-apis b. body [service api description] with apiName **service_1** c. Use **APF Certificate** 3. Provider Retrieve ACL a. Send **GET** https://{CAPIF_HOSTNAME}/access-control-policy/v1/accessControlPolicyList/${serviceApiId}?aef-id=${aefId} b. Use **serviceApiId** and **aefId** c. Use **AEF Certificate**
Expected Result	1. Response to ACL Service must accomplish: a. **404 Not Found** Response. b. body returned must accomplish **Problem Details** data structure. c. **apiInvokerPolicies** must: • status **404** • title with message "Not Found" • detail with message "No ACLs found for the requested service: {service_api_id}, aef_id: {aef_id}, invoker: {api_invoker_id} and supportedFeatures: {supported_features}". • cause with message "Wrong id".

Test Case 10: Retrieve ACL filtered by api-invoker-id not present

Test ID	capif_api_acl -10
Description	This test case will check that an API Provider get not found response if filter by not valid api-invoker-id doesn't match any registered ACL.
Pre-conditions	The Administrator must have previously registered the User The user must have the access token, the ca root certificate and the URLs to onboard an Invoker. Service published in CAPIF Security context created for Service API published
Execution Steps	1. Onboard Invoker and Provider at CCF. 2. Publish Service API with name service_1 3. Create Security Context 4. Provider Get ACL information
Information of Test	1. Onboard Invoker and provider at CCF. 2. Publish Service API at CCF:

	a. Send **POST** to ccf_publish_url https://{CAPIF_HOSTNAME}/published-apis/v1/{apfId}/service-apis b. body [service api description] with apiName **service_1** c. Use **APF Certificate** 3. Create Security Context for this Invoker a. Send **PUT** https://{CAPIF_HOSTNAME}/trustedInvokers/{apiInvokerId} b. Use **Invoker Certificate** 4. Provider Retrieve ACL a. Send **GET** https://{CAPIF_HOSTNAME}/access-control-policy/v1/accessControlPolicyList/${serviceApiId }?aef-id=${aefId}**&api-invoker-id={NOT_VALID_API_INVOKER_ID}** b. Use **serviceApiId, aefId and NOT_VALID_API_INVOKER_ID** c. Use **AEF Certificate**
Expected Result	1. Response to ACL Service must accomplish: a. **404 Not Found** Response. b. body returned must accomplish **Problem Details** data structure. c. **apiInvokerPolicies** must: • status **404** • title with message "Not Found" • detail with message "No ACLs found for the requested service: {service_api_id}, aef_id: {aef_id}, invoker: {api_invoker_id} and supportedFeatures: {supported_features}". • cause with message "Wrong id".

Test Case 11: Retrieve ACL with APF Certificate

Test ID	capif_api_acl -11
Description	This test case will check that an API Provider can't retrieve ACL from CAPIF using **APF Certificate**
Pre-conditions	The Administrator must have previously registered the User The user must have the access token, the ca root certificate and the URLs to onboard an Invoker. Service published in CAPIF Security context created for Service API published
Execution Steps	1. Onboard Invoker and Provider at CCF. 2. Publish Service API with name service_1 3. Create Security Context 4. Provider Get ACL information
Information of Test	1. Onboard Invoker and provider at CCF. 2. Publish Service API at CCF: a. Send **POST** to ccf_publish_url https://{CAPIF_HOSTNAME}/published-apis/v1/{apfId}/service-apis b. body [service api description] with apiName **service_1** c. Use **APF Certificate** 3. Create Security Context for this Invoker a. Send **PUT** https://{CAPIF_HOSTNAME}/trustedInvokers/{apiInvokerId} b. Use **Invoker Certificate** 4. Provider Retrieve ACL a. Send **GET** https://{CAPIF_HOSTNAME}/access-control-policy/v1/accessControlPolicyList/${serviceApiId}?aef-id=${aef_id} b. Use **serviceApiId** and **aefId** c. Use **APF Certificate**
Expected Result	1. Response to ACL Service must accomplish: a. **401 Unauthorized** b. Error Response Body must accomplish with **ProblemDetails** data structure with: • status 401

	• title with message "Unauthorized" • detail with message "Role not authorized for this API route". • cause with message "Certificate not authorized".

Test Case 12: Retrieve ACL with AMF Certificate

Test ID	capif_api_acl -12
Description	This test case will check that an API Provider can't retrieve ACL from CAPIF using **AMF Certificate**
Pre-conditions	The Administrator must have previously registered the User The user must have the access token, the ca root certificate and the URLs to onboard an Invoker. Service published in CAPIF Security context created for Service API published
Execution Steps	1. Onboard Invoker and Provider at CCF. 2. Publish Service API with name service_1 3. Create Security Context 4. Provider Get ACL information
Information of Test	1. Onboard Invoker and provider at CCF. 2. Publish Service API at CCF: a. Send **POST** to ccf_publish_url https://{CAPIF_HOSTNAME}/published-apis/v1/{apfId}/service-apis b. body [service api description] with apiName **service_1** c. Use **APF Certificate** 3. Create Security Context for this Invoker a. Send **PUT** https://{CAPIF_HOSTNAME}/trustedInvokers/{apiInvokerId} b. Use **Invoker Certificate** 4. Provider Retrieve ACL a. Send **GET** https://{CAPIF_HOSTNAME}/access-control-policy/v1/accessControlPolicyList/${serviceApiId}?aef-id=${aef_id} b. Use **serviceApiId** and **aefId** c. Use **AMF Certificate**
Expected Result	1. Response to ACL Service must accomplish: a. **401 Unauthorized** b. Error Response Body must accomplish with **ProblemDetails** data structure with: • status 401 • title with message "Unauthorized" • detail with message "Role not authorized for this API route". • cause with message "Certificate not authorized".

Test Case 13: Retrieve ACL with Invoker Certificate

Test ID	capif_api_acl -13
Description	This test case will check that an API Provider can't retrieve ACL from CAPIF using **Invoker Certificate**
Pre-conditions	The Administrator must have previously registered the User The user must have the access token, the ca root certificate and the URLs to onboard an Invoker. Service published in CAPIF Security context created for Service API published

Execution Steps	1. Onboard Invoker and Provider at CCF. 2. Publish Service API with name service_1 3. Create Security Context 4. Invoker Get ACL information
Information of Test	1. Onboard Invoker and provider at CCF. 2. Publish Service API at CCF: a. Send **POST** to ccf_publish_url https://{CAPIF_HOSTNAME}/published-apis/v1/{apfId}/service-apis b. body [service api description] with apiName **service_1** c. Use **APF Certificate** 3. Create Security Context for this Invoker a. Send **PUT** https://{CAPIF_HOSTNAME}/trustedInvokers/{apiInvokerId} b. Use **Invoker Certificate** 4. Invoker Retrieve ACL a. Send **GET** https://{CAPIF_HOSTNAME}/access-control-policy/v1/accessControlPolicyList/${serviceApiId}?aef-id=${aef_id} b. Use **serviceApiId** and **aefId** c. Use **Invoker Certificate**
Expected Result	1. Response to ACL Service must accomplish: a. **401 Unauthorized** b. Error Response Body must accomplish with **ProblemDetails** data structure with: • status 401 • title with message "Unauthorized" • detail with message "Role not authorized for this API route". • cause with message "Certificate not authorized".

Test Case 14: No ACL for invoker after being removed

Test ID	capif_api_acl -14
Description	This test case will check that ACLs are removed after invoker is removed.
Pre-conditions	The Administrator must have previously registered the User The user must have the access token, the ca root certificate and the URLs to onboard an Invoker. Service published in CAPIF Security context created for Service API published
Execution Steps	1. Onboard Invoker and Provider at CCF. 2. Publish Service API with name service_1 3. Create Security Context 4. Provider Get ACL information 5. Remove Invoker from CAPIF 6. Provider Get ACL information
Information of Test	1. Onboard Invoker and provider at CCF. 2. Publish Service API at CCF: a. Send **POST** to ccf_publish_url https://{CAPIF_HOSTNAME}/published-apis/v1/{apfId}/service-apis b. body [service api description] with apiName **service_1** c. Use **APF Certificate** 3. Create Security Context for this Invoker a. Send **PUT** https://{CAPIF_HOSTNAME}/trustedInvokers/{apiInvokerId} b. Use **Invoker Certificate** 4. Provider Retrieve ACL a. Send **GET** https://{CAPIF_HOSTNAME}/access-control-policy/v1/accessControlPolicyList/${serviceApiId}?aef-id=${aef_id}&api-invoker-id={api-invoker-id} b. Use **serviceApiId**, **aefId and api-invoker-id** c. Use **AEF Certificate** 5. Remove Invoker from CAPIF

	6. Provider Retrieve ACL a. Send **GET** https://{CAPIF_HOSTNAME}/access-control-policy/v1/accessControlPolicyList/${serviceApiId}?aef-id=${aef_id}&api-invoker-id={api-invoker-id} b. Use **serviceApiId**, **aefId and api-invoker-id** c. Use **AEF Certificate**
Expected Result	1. Response to first ACL Service request must accomplish: a. **200 OK** Response. b. body returned must accomplish **AccessControlPolicyList** data structure. c. **apiInvokerPolicies** must: • contain only one object. • apiInvokerId must match with apiInvokerId registered previously 2. Response to second ACL Service request must accomplish: a. **404 Not Found** Response. b. body returned must accomplish **Problem Details** data structure. c. **apiInvokerPolicies** must: • status **404** • title with message "Not Found" • detail with message "No ACLs found for the requested service: {NOT_VALID_SERVICE_API_ID}, aef_id: {AEF_ID_NOT_VALID}, invoker: None and supportedFeatures: None". • cause with message "Wrong id".

D.8 Test Plan for CAPIF API Logging Service

This section list test cases for the CAPIF_Logging_API_Invocation_API.

Test Case 1: Creates a new individual CAPIF Log Entry

Test ID	capif_api_logging -1
Description	This test case will check that a CAPIF AEF can create log entry to Logging Service
Pre-conditions	The Administrator must have previously registered the User The user must have the access token, the ca root certificate and the URLs to onboard an Invoker. Service published in CAPIF
Execution Steps	1. Onboard Invoker and Provider at CCF. 2. Publish Service API 3. Create Log Entry
Information of Test	1. Onboard Invoker and provider at CCF. 2. Publish Service API at CCF: a. Send **POST** to ccf_publish_url https://{CAPIF_HOSTNAME}/published-apis/v1/{apfId}/service-apis b. body [service api description] with apiName **service_1** c. Use **APF Certificate** 3. Log Entry: a. Send **POST** to https://{CAPIF_HOSTNAME}/api-invocation-logs/v1/{aefId}/logs b. Body **InvocationLog** c. Use **AEF Certificate**
Expected Result	1. Response to Logging Service must accomplish: a. **201 Created** b. Response Body must follow **InvocationLog** data structure with: • aefId • apiInvokerId • logs c. Response Header **Location** must be received with URI to new resource created, following this structure: **{apiRoot}/api-invocation-logs/v1/{aefId}/logs/{logId}**

Test Case 2: Creates a new individual CAPIF Log Entry with invalid aefId

Test ID	capif_api_logging -2
Description	This test case will check that a CAPIF subscriber (AEF) cannot create Log Entry without valid aefId
Pre-conditions	The Administrator must have previously registered the User The user must have the access token, the ca root certificate and the URLs to onboard an Invoker. Service published in CAPIF
Execution Steps	1. Onboard Invoker and Provider at CCF. 2. Publish Service API 3. Create Log Entry
Information of Test	1. Onboard Invoker and provider at CCF. 2. Publish Service API at CCF: a. Send **POST** to ccf_publish_url https://{CAPIF_HOSTNAME}/published-apis/v1/{apfId}/service-apis b. body [service api description] with apiName **service_1** c. Use **APF Certificate** 3. Log Entry: a. Send **POST** to https://{CAPIF_HOSTNAME}/api-invocation-logs/v1/{not-valid-aefId}/logs b. **Body InvocationLog** c. Use **AEF Certificate**
Expected Result	1. Response to Logging Service must accomplish: a. **404 Not Found** b. Error Response Body must accomplish with **ProblemDetails** data structure with: • status 404 • title with message "Not Found" • detail with message "Exposer not exist". • cause with message "Exposer id not found".

Test Case 3: Creates a new individual CAPIF Log Entry with invalid serviceAPI

Test ID	capif_api_logging-3
Description	This test case will check that a CAPIF subscriber (AEF) cannot create Log Entry without valid aefId
Pre-conditions	The Administrator must have previously registered the User The user must have the access token, the ca root certificate and the URLs to onboard an Invoker. Service published in CAPIF
Execution Steps	1. Onboard Invoker and Provider at CCF. 2. Publish Service API 3. Create Log Entry
Information of Test	1. Onboard Invoker and provider at CCF. 2. Publish Service API at CCF: a. Send **POST** to ccf_publish_url https://{CAPIF_HOSTNAME}/published-apis/v1/{apfId}/service-apis b. body [service api description] with apiName **service_1** c. Use **APF Certificate** 3. Log Entry: a. Send **POST** to https://{CAPIF_HOSTNAME}/api-invocation-logs/v1/{aefId}/logs b. **Body InvocationLog with serviceAPI apiName apiId not valid]** c. Use **AEF Certificate**

Expected Result	1. Response to Logging Service must accomplish: a. **404 Not Found** b. Error Response Body must accomplish with **ProblemDetails** data structure with: • status 404 • title with message "Not Found" • detail with message "Service not exist". • cause with message "Service id not found".

Test Case 4: Creates a new individual CAPIF Log Entry with invalid apiInvokerId

Test ID	capif_api_logging-4
Description	This test case will check that a CAPIF subscriber (AEF) cannot create Log Entry without valid apiInvokerId
Pre-conditions	The Administrator must have previously registered the User The user must have the access token, the ca root certificate and the URLs to onboard an Invoker. Service published in CAPIF
Execution Steps	1. Onboard Invoker and Provider at CCF. 2. Publish Service API 3. Create Log Entry
Information of Test	1. Onboard Invoker and provider at CCF. 2. Publish Service API at CCF: a. Send **POST** to ccf_publish_url https://{CAPIF_HOSTNAME}/published-apis/v1/{apfId}/service-apis b. body serviceApiDescription with apiName **service_1** c. Use **APF Certificate** 3. Log Entry: a. Send **POST** to https://{CAPIF_HOSTNAME}/api-invocation-logs/v1/{aefId}/logs b. **Body InvocationLog with apiInvokerId not valid** c. Use **AEF Certificate**
Expected Result	1. Response to Logging Service must accomplish: a. **404 Not Found** b. Error Response Body must accomplish with **ProblemDetails** data structure with: • status 404 • title with message "Not Found" • detail with message "Invoker not exist". • cause with message "Invoker id not found".

Test Case 5: Creates a new individual CAPIF Log Entry with invalid aefId in body

Test ID	capif_api_logging-5
Description	This test case will check that a CAPIF subscriber (AEF) cannot create Log Entry without valid aefId in body
Pre-conditions	The Administrator must have previously registered the User The user must have the access token, the ca root certificate and the URLs to onboard an Invoker. Service published in CAPIF
Execution Steps	1. Onboard Invoker and Provider at CCF. 2. Publish Service API 3. Create Log Entry

Information of Test	1. Onboard Invoker and provider at CCF. 2. Publish Service API at CCF: a. Send **POST** to ccf_publish_url https://{CAPIF_HOSTNAME}/published-apis/v1/{apfId}/service-apis b. body serviceApiDescription with apiName **service_1** c. Use **APF Certificate** 3. Log Entry: a. Send **POST** to https://{CAPIF_HOSTNAME}/api-invocation-logs/v1/{aefId}/logs b. **Body InvocationLog with aefId not valid** c. Use **AEF Certificate**
Expected Result	1. Response to Logging Service must accomplish: a. **401 Unauthorized** b. Error Response Body must accomplish with **ProblemDetails** data structure with: • status 401 • title with message "Unauthorized" • detail with message "AEF id not matching in request and body". • cause with message “Not identical AEF id".

D.9 Test Plan for CAPIF API Events Service

This section list test cases for the CAPIF_Events_API.

Test Case 1: Creates a New Individual CAPIF Event Subscription

Test ID	capif_api_events-1
Description	This test case will check that a CAPIF subscriber (Invoker or Provider) can Subscribe to Events
Pre-conditions	The Administrator must have previously registered the User The user must have the access token, the ca root certificate and the URLs to onboard an Invoker or Provider.
Execution Steps	1. Onboard Invoker at CCF. 2. Subscribe to Events 3. Retrieve (subscribeId) and (subscriptionId) from Location Header
Information of Test	1. Onboard Invoker at CCF. 2. Event Subscription: a. Send POST to **https://{CAPIF_HOSTNAME}/capif-events/v1/{subscriberId}/subscriptions** b. Use Invoker **Certificate**
Expected Result	1. Onboard Invoker. 2. Response to Event Subscription must accomplish: a. Status code 201 Created b. The URI of the created resource shall be returned in the “Location” HTTP header, following this structure {**apiRoot}/capif-events/{apiVersion}/{subscriberId}/subscriptions/{subscriptionId}** c. Response body must follow EvenstSubscription data structure: 3. Event subscriptions are stored in CAPIF Database

Test Case 2: Creates a New Individual CAPIF Event Subscription with invalid subscriberId

Test ID	capif_api_events-2

Description	This test case will check that a CAPIF subscriber (Invoker or Publisher) cannot Subscribe to Events without valid SubcriberId
Pre-conditions	The Administrator must have previously registered the User The user must have the access token, the ca root certificate and the URLs to onboard an Invoker or Provider.
Execution Steps	1. Onboard Invoker at CCF. 2. Subscribe to Events with an invalid subscriberId 3. Retrieve (subscribeId) and (subscriptionId) from Location Header
Information of Test	1. Onboard Invoker at CCF. 2. Event Subscription: a. Send POST to **https://{CAPIF_HOSTNAME}/capif-events/v1/{INVALID_SUBSCRIBER_ID}/subscriptions** b. Use I**nvoker Certificate**
Expected Result	1. Onboard Invoker. 2. Response to Event Subscription must accomplish: a. Status code 4**04 Not Found** b. Error Response Body must accomplish with ProblemDetails data structure with: • Status 404 • Title with message “Not Found” • Title with message “Invoker or APF or AEF or AMF Not found” • Detail with message “Subscriber Not Found” 3. Event subscription are not stored in CAPIF Database

Test Case 3: Delete an Individual CAPIF Event Subscription

Test ID	capif_api_events-3
Description	This test case will check that a CAPIF subscriber (Invoker or Publisher) can Delete an Event Subscription
Pre-conditions	The Administrator must have previously registered the User The user must have the access token, the ca root certificate and the URLs to onboard an Invoker or Provider.
Execution Steps	1. Onboard Invoker at CCF. 2. Subscribe to Events with an invalid subscriberId 3. Retrieve (subscribeId) and (subscriptionId) from Location Header 4. Remove Event Subscription
Information of Test	1. Onboard Invoker at CCF. 2. Event Subscription: a. Send POST to **https://{CAPIF_HOSTNAME}/capif-events/v1/{subscriberId}/subscriptions** b. Use I**nvoker Certificate** 3. Remove Event Subscription: a. Send DELETE to **https://{CAPIF_HOSTNAME}/capif-events/v1/{subscriberId}/subscriptions/{subscriptionId}** b. Use Invoker Certificate
Expected Result	1. Onboard Invoker. 2. Response to Event Subscription must accomplish: a. Status code 4**04 Not Found** b. Error Response Body must accomplish with ProblemDetails data structure with: • Status 404 • Title with message “Not Found” • Title with message “Invoker or APF or AEF or AMF Not found” • Detail with message “Subscriber Not Found” 3. Event subscriptions are stored in CAPIF Database 4. Remove Event Subscription: a. 204 No Content

	5. Event Subscription is not present in CAPIF Database

Test Case 4: Delete an Individual CAPIF Event Subscription with an Invalid subscriberId

Test ID	capif_api_events-4
Description	This test case will check that a CAPIF subscriber (Invoker or Publisher) cannot Delete to Events without valid SubcriberId
Pre-conditions	The Administrator must have previously registered the User The user must have the access token, the ca root certificate and the URLs to onboard an Invoker or Provider. CAPIF subscriber is subscribed to Events.
Execution Steps	1. Onboard Invoker at CCF. 2. Subscribe to Events with an invalid subscriberId 3. Retrieve (subscribeId) and (subscriptionId) from Location Header 4. Remove Event Subscription with not valid Subscriber
Information of Test	1. Onboard Invoker at CCF. 2. Event Subscription: a. Send POST to **https://{CAPIF_HOSTNAME}/capif-events/v1/{subscriberId}/subscriptions** b. Use **I**nvoker **Certificate** 3. Remove Event Subscription with Invalid subscriberId: a. Send DELETE to **https://{CAPIF_HOSTNAME}/capif-events/v1/{INVALID_SUBSCRIBER_ID}/subscriptions/{subscriptionId}** b. Use Invoker Certificate
Expected Result	1. Onboard Invoker. 2. Response to Event Subscription must accomplish: a. Status code **404 Not Found** b. Error Response Body must accomplish with ProblemDetails data structure with: • Status 404 • Title with message "Not Found" • Title with message "Invoker or APF or AEF or AMF Not found" • Detail with message "Subscriber Not Found" 3. Event subscriptions are stored in CAPIF Database 4. Remove Event Subscription with invalid subscriberId: a. Status code **404 Not Found** b. Error Response Body must accomplish with ProblemDetails data structure with: • Status 404 • Title with message "Not Found" • Title with message "Invoker or APF or AEF or AMF Not found" • Detail with message "Subscriber Not Found"

Test Case 5: Delete an Individual CAPIF Event Subscription with an Invalid subscriptionId

Test ID	capif_api_events-5
Description	This test case will check that a CAPIF subscriber (Invoker or Publisher) cannot Delete an Event Subscription without valid SubscriptionId
Pre-conditions	The Administrator must have previously registered the User The user must have the access token, the ca root certificate and the URLs to onboard an Invoker or Provider.

Execution Steps	1. Onboard Invoker at CCF. 2. Subscribe to Events with an invalid subscriberId 3. Retrieve (subscribeId) and (subscriptionId) from Location Header 4. Remove Event Subscription with not valid subscriptionId

Information of Test	1. Onboard Invoker at CCF.
	2. Event Subscription:
	  a. Send POST to **https://{CAPIF_HOSTNAME}/capif-events/v1/{subscriberId}/subscriptions**
	  b. Use Invoker **Certificate**
	3. Remove Event Subscription with not valid subscriptionId:
	  a. Send DELETE to **https://{CAPIF_HOSTNAME}/capif-events/v1/{subscriberId}/subscriptions/{SUBSCRIPTION_ID_NOT_VALID}**
	  b. Use Invoker Certificate

Expected Result	1. Onboard Invoker. 2. Response to Event Subscription must accomplish: a. Status code **404 Not Found** b. Error Response Body must accomplish with ProblemDetails data structure with: • Status 404 • Title with message "Not Found" • Title with message "Invoker or APF or AEF or AMF Not found" • Detail with message "Subscriber Not Found" 3. Event subscriptions are stored in CAPIF Database 4. Remove Event Subscription: a. Status code **404 Not Found** b. Error Response Body must accomplish with ProblemDetails data structure with: • Status 404 • Title with message "Not Found" • Title with message "Service API not existing" • Detail with message "Event API subscription Id not found"

Test Case 6: Invoker receives Service API invocation Events

Test ID	capif_api_events-6
Description	This test case will check that a CAPIF Invoker subscribed to SERVICE_API_INVOCATION_SUCCESS and SERVICE_API_INVOCATION_FAILURE, receive the notification when AEF Send **TO** logging service result of invocations to their APIs.
Pre-conditions	The Administrator must have previously registered the User The user must have the access token, the ca root certificate and the URLs to onboard an Invoker and Provider. Provider had a Service API published on CAPIF Mock Server is up and running to receive requests. Mock Server is clean
Execution Steps	1. Onboard Provider and publish one API at CCF 2. Onboard Invoker at CCF. 3. Discover Published APIs and extract apiIds and apiNames. 4. Subscribe **SERVICE_API_INVOCATION_SUCCESS** and **SERVICE_API_INVOCATION_FAILURE** event filtering by aefId. 5. Retrieve (subscribeId) and (subscriptionId) from Location Header 6. Emulate Success and Failure on API invocation of provider by Invoker, using Invocation Logs API.

Information of Test	1. Onboard Provider
	2. Publish Service API at CCF:
	  a. Send POST to **https://{CAPIF_HOSTNAME}/published-apis/v1/{apfId}/service-apis**
	  b. Body serviceAPIDescription with apiName service_1
	  c. Get apiId
	  d. Use **APF Certificate**
	3. Onboard Invoker at CCF.

	4. Discover published APIs: a. Get **Api Ids** And **Api Names** from response. 5. Event Subscription to SERVICE_API_INVOCATION_SUCCESS and SERVICE_API_INVOCATION_FAILURE of provider previously registered: a. Send POST to **https://{CAPIF_HOSTNAME}/capif-events/v1/{subscriberId}/subscriptions** b. Body of Event Subscription request with: • Events: **['SERVICE_API_INVOCATION_SUCCESS','SERVICE_API_INVOCATION_FAILURE']** • eventFilter: only receive events from provider's aefId. c. Use Invoker **Certificate** 6. Create a Log Entry emulating provider receive Success and Failure api Invocation from Invoker a. Send **POST** to **https://{CAPIF_HOSTNAME}/api-invocation-logs/v1/{aefId}/logs** b. body log entry request body with: • aefId from provider published. • apiInvokerId from invoker onboarded. • apiId of published API • apiName of published API • 200 and 400 results in two logs. c. Use **AEF Certificate**
Expected Result	1. Response to Event Subscription must accomplish: a. **201 Created** b. The URI of the created resource shall be returned in the "Location" HTTP header, following this structure: **{apiRoot}/capif-events/{apiVersion}/{subscriberId}/subscriptions/{subscriptionId}** c. Response Body must follow **EventSubscription** data structure. 2. Response to creation of log entry on CCF must accomplish: a. **201 Created** b. The URI of the created resource shall be returned in the "Location" HTTP header, following this structure: **{apiRoot}/api-invocation-logs/{apiVersion}/{aefId}/subscriptions/{logId}** 3. Mock Server received messages must accomplish: a. **Two Events have been received**. b. Validate received events follow **EventNotification** data structure, with **invocationLog** in **eventDetail** parameter. • One should be **SERVICE_API_INVOCATION_SUCCESS** related with **200** result at Log. • The other one must be **SERVICE_API_INVOCATION_FAILURE** related with **400** result at Log.

Test Case 7: Invoker subscribe to Service API Available and Unavailable events

Test ID	capif_api_events-7
Description	This test case will check that a CAPIF Invoker subscribed to SERVICE_API_AVAILABLE and SERVICE_API_UNAVAILABLE, receive the notification when AEF publish and remove it.
Pre-conditions	The Administrator must have previously registered the User The user must have the access token, the ca root certificate and the URLs to onboard an Invoker and Provider. Provider had a Service API published on CAPIF Mock Server is up and running to receive requests. Mock Server is clean

Execution Steps	1. Onboard Provider and publish one API at CCF 2. Onboard Invoker at CCF. 3. Discover Published APIs and extract apiIds and apiNames. 4. Subscribe **SERVICE_API_INVOCATION_SUCCESS** and **SERVICE_API_INVOCATION_FAILURE** event filtering by aefId. 5. Retrieve (subscribeId) and (subscriptionId) from Location Header 6. Provider publish new API 7. Provider remove published API
Information of Test	1. Perform provider registration 2. Publish Service API at CCF: a. Send **POST** to ccf_publish_url **https://{CAPIF_HOSTNAME}/published-apis/v1/{apfId}/service-apis** b. body [service api description] with apiName **service_1** c. Store **serviceApiId** d. Use **APF Certificate** 3. Perform invoker onboarding 4. Discover published APIs: a. Get **Api Ids** And **Api Names** from response. 5. Event Subscription to SERVICE_API_AVAILABLE and SERVICE_API_UNAVAILABLE of provider previously registered: a. Send **POST** to **https://{CAPIF_HOSTNAME}/capif-events/v1/{subscriberId}/subscriptions** b. body event subscription request body with: • events: **['SERVICE_API_AVAILABLE','SERVICE_API_UNAVAILABLE']** • eventFilter: only receive events from provider's aefId. c. Use **Invoker Certificate** 6. Publish new Service API at CCF: a. Send **POST** to ccf_publish_url **https://{CAPIF_HOSTNAME}/published-apis/v1/{apfId}/service-apis** b. body [service api description] with apiName **service_2** c. Store **serviceApiId** d. Use **APF Certificate** 7. Remove published Service API at CCF: a. Send **DELETE** to resource URL **https://{CAPIF_HOSTNAME}/published-apis/v1/{apfId}/service-apis/{SERVICE_API_ID}** b. Use **APF Certificate**
Expected Result	1. Response to Event Subscription must accomplish: a. **201 Created** b. The URI of the created resource shall be returned in the "Location" HTTP header, following this structure: **{apiRoot}/capif-events/{apiVersion}/{subscriberId}/subscriptions/{subscriptionId}** c. Response Body must follow **EventSubscription** data structure. 2. Mock Server received messages must accomplish: a. **Two Events have been received**. b. Validate received events follow **EventNotification** data structure, with **invocationLog** in **eventDetail** parameter. • One should be **SERVICE_API_AVAILABLE** apiId of **service_2** published API. • The other one must be **SERVICE_API_UNAVAILABLE** apiId of **service_1** published API.

Test Case 8: Invoker subscribe to Service API Update

Test ID	capif_api_events-8
Description	This test case will check that a CAPIF Invoker subscribed to SERVICE_API_UPDATE, receive the notification when AEF Update some information on API Published.

Pre-conditions	The Administrator must have previously registered the User The user must have the access token, the ca root certificate and the URLs to onboard an Invoker and Provider. Provider had a Service API published on CAPIF Mock Server is up and running to receive requests. Mock Server is clean
Execution Steps	1. Onboard Provider and publish one API at CCF 2. Onboard Invoker at CCF. 3. Discover Published APIs and extract apiIds and apiNames. 4. Subscribe to **SERVICE_API_UPDATE** event filtering by aefId. 5. Retrieve (subscribeId) and (subscriptionId) from Location Header 6. Provider update information of Service API Published.
Information of Test	1. Perform provider registration 2. Publish Service API at CCF: a. Send **POST** to ccf_publish_url **https://{CAPIF_HOSTNAME}/published-apis/v1/{apfId}/service-apis** b. body [service api description] with apiName **service_1** c. Store **serviceApiId** d. Use **APF Certificate** 3. Perform invoker onboarding 4. Discover published APIs: a. Get **Api Ids** And **Api Names** from response. 5. Event Subscription to SERVICE_API_AVAILABLE and SERVICE_API_UNAVAILABLE of provider previously registered: a. Send **POST** to **https://{CAPIF_HOSTNAME}/capif-events/v1/{subscriberId}/subscriptions** b. body event subscription request body with: • events: **['SERVICE_API_UPDATE']** • eventFilter: only receive events from provider's aefId. c. Use **Invoker Certificate** 6. Update published API at CCF: a. Send **PUT** to resource URL https://{CAPIF_HOSTNAME}/published-apis/v1/{apfId}/service-apis/{serivceApiId} b. body [service api description] with overrided **apiName** to **service_1**_modified** c. Use **APF Certificate**
Expected Result	1. Response to Event Subscription must accomplish: a. **201 Created** b. The URI of the created resource shall be returned in the "Location" HTTP header, following this structure: **{apiRoot}/capif-events/{apiVersion}/{subscriberId}/subscriptions/{subscriptionId}** c. Response Body must follow **EventSubscription** data structure. 2. Response to Update Published Service API: a. **200 OK** b. Response Body must follow **ServiceAPIDescription** data structure with: • apiName **service_1**_modified** 3. Mock Server received messages must accomplish: a. **One Events have been received**. b. Validate received events follow **EventNotification** data structure, with **invocationLog** in **eventDetail** parameter. • Event should be **SERVICE_API_UPDATE** with **eventDetail** with modified **apiName**.

Test Case 9: Provider subscribe to API Invoker Events

Test ID	capif_api_events-9
Description	This test case will check that a CAPIF Provider subscribed to API Invoker events (API_INVOKER_ONBOARDED, API_INVOKER_UPDATED and

	API_INVOKER_OFFBOARDED), receive the notifications when Invoker is onboarded, updated and removed respectively.
Pre-conditions	The Administrator must have previously registered the User The user must have the access token, the ca root certificate and the URLs to onboard an Invoker and Provider. Provider had a Service API published on CAPIF Mock Server is up and running to receive requests. Mock Server is clean
Execution Steps	1. Onboard Provider and publish one API at CCF 2. Subscribe Provider to **API_INVOKER_ONBOARDED, API_INVOKER_UPDATED and API_INVOKER_OFFBOARDED** events. 3. Onboard Invoker at CCF. 4. Update Onboarding Information at CCF with a minor change on "notificationDestination" 5. Offboard Invoker
Information of Test	1. Perform provider registration 2. Event Subscription to API_INVOKER_ONBOARDED, API_INVOKER_UPDATED and API_INVOKER_OFFBOARDED events: a. Send **POST** to **https://{CAPIF_HOSTNAME}/capif-events/v1/{subscriberId}/subscriptions** b. body event subscription request body with: • events: **['API_INVOKER_ONBOARDED', 'API_INVOKER_UPDATED', 'API_INVOKER_OFFBOARDED']** c. Use **Provider AMF Certificate** 3. Perform invoker onboarding 4. Update information of previously onboarded Invoker: a. Send **PUT** to **https://{CAPIF_HOSTNAME}/api-invoker-management/v1/onboardedInvokers/{onboardingId}** b. Reference Request Body is: [put invoker onboarding body] c. "notificationDestination": "**http://host.docker.internal:8086/netapp_new_callback**", 5. Offboard: a. Send **DELETE** to **https://{CAPIF_HOSTNAME}/api-invoker-management/v1/onboardedInvokers/{onboardingId}**
Expected Result	1. Response to Event Subscription must accomplish: a. **201 Created** b. The URI of the created resource shall be returned in the "Location" HTTP header, following this structure: **{apiRoot}/capif-events/{apiVersion}/{subscriberId}/subscriptions/{subscriptionId}** c. Response Body must follow **EventSubscription** data structure. 2. Response to Onboard request must accomplish: a. **201 Created** b. Response Body must follow **APIInvokerEnrolmentDetails** data structure with: • apiInvokerId • onboardingInformation->apiInvokerCertificate must contain the public key signed. c. Response Header **Location** must be received with URI to new resource created, following this structure: **{apiRoot}/api-invoker-management/{apiVersion}/onboardedInvokers/{onboardingId}** 3. Response to Update Request (PUT) with minor change must contain: a. **200 OK** response. b. notificationDestination on response must contain the new value 4. Response to Offboard Request (DELETE) must contain: a. **204 No Content** 5. Mock Server received messages must accomplish: a. **Three Events have been received**. b. Validate received events follow **EventNotification** data structure, with **apiInvokerIds** in **eventDetail** parameter. • One Event should be **API_INVOKER_ONBOARDED** with **eventDetail** with modified **apiInvokerId**.

	• One Event should be **API_INVOKER_UPDATED** with **eventDetail** with modified **apiInvokerId**. • One Event should be **API_INVOKER_OFFBOARDED** with **eventDetail** with modified **apiInvokerId**.

Test Case 10: Provider subscribe to ACL Update event

Test ID	capif_api_events-10
Description	This test case will check that a CAPIF Provider subscribed to ACCESS_CONTROL_POLICY_UPDATE receive a notification when ACL Changes.
Pre-conditions	The Administrator must have previously registered the User The user must have the access token, the ca root certificate and the URLs to onboard an Invoker and Provider. Provider had a Service API published on CAPIF API Invoker had a Security Context for the Service API published by Provider Mock Server is up and running to receive requests. Mock Server is clean
Execution Steps	1. Onboard Provider. 2. Publish a provider API with name **service_1**. 3. Onboard Invoker at CCF. 4. Subscribe Provider to **ACCESS_CONTROL_POLICY_UPDATE** event. 5. Discover APIs filtered by **aef_id** 6. Create Security Context for Invoker. 7. Provider Retrieve ACL
Information of Test	1. Perform provider registration 2. Perform invoker onboarding 3. Event Subscription to **ACCESS_CONTROL_POLICY_UPDATE** event: a. Send **POST** to https://{CAPIF_HOSTNAME}/capif-events/v1/{subscriberId}/subscriptions b. body event subscription request body with: • events: **['ACCESS_CONTROL_POLICY_UPDATE']** • eventFilters: apiInvokerIds array with apiInvokerId of invoker c. Use **Provider AMF Certificate** 4. Discover published APIs 5. Create Security Context for Invoker a. Send **PUT** https://{CAPIF_HOSTNAME}/trustedInvokers/{apiInvokerId} b. Use **Invoker Certificate** 6. Provider Retrieve ACL a. Send **GET** https://{CAPIF_HOSTNAME}/access-control-policy/v1/accessControlPolicyList/${serviceApiId}?aef-id=${aef_id} b. Use **serviceApiId** and **aefId** c. Use AEF Provider Certificate
Expected Result	1. Response to Event Subscription must accomplish: a. **201 Created** b. The URI of the created resource shall be returned in the "Location" HTTP header, following this structure: **{apiRoot}/capif-events/{apiVersion}/{subscriberId}/subscriptions/{subscriptionId}** c. Response Body must follow **EventSubscription** data structure. 2. Create security context: a. body returned must accomplish **ServiceSecurity** data structure. b. Location Header must contain the new resource URL **{apiRoot}/capif-security/v1/trustedInvokers/{apiInvokerId}** 3. ACL Response: a. **200 OK** Response. b. body returned must accomplish **AccessControlPolicyList** data structure. c. apiInvokerPolicies must:

	• contain only one object. • apiInvokerId must match apiInvokerId registered previously. 4. Mock Server received messages must accomplish: a. **One Event has been received**. b. Validate received event follow **EventNotification** data structure, with **accCtrlPolListExt** in **eventDetail** parameter. • One Event should be **ACCESS_CONTROL_POLICY_UPDATE** with **eventDetail** with **accCtrlPolListExt** including the **apiId** and **apiInvokerPolicies**.

Test Case 11: Provider receives an ACL unavailable event when invoker removes Security Context

Test ID	capif_api_events-11
Description	This test case will check that a CAPIF Invoker subscribed to ACCESS_CONTROL_POLICY_UNAVAILABLE will receive the notification when AEF remove Security Context created previously.
Pre-conditions	The Administrator must have previously registered the User The user must have the access token, the ca root certificate and the URLs to onboard an Invoker and Provider. Provider had a Service API published on CAPIF API Invoker had a Security Context for the Service API published by Provider Mock Server is up and running to receive requests. Mock Server is clean
Execution Steps	1. Onboard Provider. 2. Publish a provider API with name **service_1**. 3. Onboard Invoker at CCF. 4. Subscribe Invoker to **ACCESS_CONTROL_POLICY_UNAVAILABLE** event. 5. Discover APIs filtered by **aef_id** 6. Create Security Context for Invoker. 7. Provider Retrieve ACL 8. Remove Security Context for Invoker.
Information of Test	1. Perform provider registration 2. Perform invoker onboarding 3. Event Subscription to **ACCESS_CONTROL_POLICY_UNAVAILABLE** event: a. Send **POST** to https://{CAPIF_HOSTNAME}/capif-events/v1/{subscriberId}/subscriptions b. body event subscription request body with: • events: **['ACCESS_CONTROL_POLICY_UNAVAILABLE']** • eventFilters: apiInvokerIds array with apiInvokerId of invoker c. Use **Invoker Certificate** 4. Discover published APIs 5. Create Security Context for Invoker a. Send **PUT** https://{CAPIF_HOSTNAME}/trustedInvokers/{apiInvokerId} b. Use **Invoker Certificate** 6. Provider Retrieve ACL a. Send **GET** https://{CAPIF_HOSTNAME}/access-control-policy/v1/accessControlPolicyList/${serviceApiId}?aef-id=${aef_id} b. Use **serviceApiId** and **aefId** c. Use AEF Provider Certificate 7. Delete Security Context of Invoker by Provider: a. Send **DELETE** https://{CAPIF_HOSTNAME}/trustedInvokers/{apiInvokerId} b. Use **AEF Certificate**
Expected Result	1. Response to Event Subscription must accomplish: a. **201 Created**

	b. The URI of the created resource shall be returned in the "Location" HTTP header, following this structure: **{apiRoot}/capif-events/{apiVersion}/{subscriberId}/subscriptions/{subscriptionId}** c. Response Body must follow **EventSubscription** data structure. 2. Create security context: a. body returned must accomplish **ServiceSecurity** data structure. b. Location Header must contain the new resource URL **{apiRoot}/capif-security/v1/trustedInvokers/{apiInvokerId}** 3. ACL Response: a. **200 OK** Response. b. body returned must accomplish **AccessControlPolicyList** data structure. c. apiInvokerPolicies must: • contain only one object. • apiInvokerId must match apiInvokerId registered previously. 4. Delete security context: a. **204 No Content** response 5. Mock Server received messages must accomplish: a. **One Event has been received**. b. Validate received event follow **EventNotification** data structure, with **accCtrlPolListExt** in **eventDetail** parameter. • One Event should be **ACCESS_CONTROL_POLICY_UNAVAILABLE** without **eventDetail**.

Test Case 12: invoker receives an invoker Authorization Revoked and ACL unavailable event when Provider revoke Invoker Authorization

Test ID	capif_api_events-12
Description	This test case will check that a CAPIF Invoker subscribed to API_INVOKER_AUTHORIZATION_REVOKED and ACCESS_CONTROL_POLICY_UNAVAILABLE receive both notification when AEF revoke invoker's authorization.
Pre-conditions	The Administrator must have previously registered the User The user must have the access token, the ca root certificate and the URLs to onboard an Invoker and Provider. Provider had a Service API published on CAPIF API Invoker had a Security Context for the Service API published by Provider Mock Server is up and running to receive requests. Mock Server is clean
Execution Steps	1. Onboard Provider. 2. Publish a provider API with name **service_1**. 3. Onboard Invoker at CCF. 4. Invoker to **ACCESS_CONTROL_POLICY_UNAVAILABLE and API_INVOKER_AUTHORIZATION_REVOKED** events. 5. Discover APIs filtered by **aef_id** 6. Create Security Context for Invoker. 7. Revoke Authorization by Provider
Information of Test	1. Perform provider registration 2. Perform invoker onboarding 3. Event Subscription to **ACCESS_CONTROL_POLICY_UNAVAILABLE and API_INVOKER_AUTHORIZATION_REVOKED** event: a. Send **POST** to https://{CAPIF_HOSTNAME}/capif-events/v1/{subscriberId}/subscriptions b. body event subscription request body with: • events: **['ACCESS_CONTROL_POLICY_UNAVAILABLE','API_INVOKER_AUTHORIZATION_REVOKED']** • eventFilters: apiInvokerIds array with apiInvokerId of invoker c. Use **Invoker Certificate** 4. Discover published APIs

	5. Create Security Context for Invoker a. Send **PUT** https://{CAPIF_HOSTNAME}/trustedInvokers/{apiInvokerId} b. Use **Invoker Certificate** 6. Provider Retrieve ACL a. Send **GET** https://{CAPIF_HOSTNAME}/access-control-policy/v1/accessControlPolicyList/${serviceApiId}?aef-id=${aef_id} b. Use **serviceApiId** and **aefId** c. Use AEF Provider Certificate 7. Revoke Authorization by Provider: a. Send **POST** https://{CAPIF_HOSTNAME}/trustedInvokers/{apiInvokerId}/delete b. body security notification body c. Using **AEF Certificate**.
Expected Result	1. Response to Event Subscription must accomplish: a. **201 Created** b. The URI of the created resource shall be returned in the "Location" HTTP header, following this structure: **{apiRoot}/capif-events/{apiVersion}/{subscriberId}/subscriptions/{subscriptionId}** c. Response Body must follow **EventSubscription** data structure. 2. Create security context: a. body returned must accomplish **ServiceSecurity** data structure. b. Location Header must contain the new resource URL **{apiRoot}/capif-security/v1/trustedInvokers/{apiInvokerId}** 3. ACL Response: a. **200 OK** Response. b. body returned must accomplish **AccessControlPolicyList** data structure. c. apiInvokerPolicies must: • contain only one object. • apiInvokerId must match apiInvokerId registered previously. 4. Revoke Authorization: a. **204 No Content** response 5. Mock Server received messages must accomplish: a. **Two Event has been received**. b. Validate received event follow **EventNotification** data structure, with **accCtrlPolListExt** in **eventDetail** parameter. • One Event should be **ACCESS_CONTROL_POLICY_UNAVAILABLE** without **eventDetail**. • One Event should be **API_INVOKER_AUTHORIZATION_REVOKED** without **eventDetail**.

D.10 Test Plan for CAPIF API Auditing Service

This section list test cases for the CAPIF_Auditing_API.

Test Case 1: Get CAPIF Log Entry

Test ID	capif_api_auditing-1
Description	This test case will check that a CAPIF AMF can get log entry to Logging Service
Pre-conditions	The Administrator must have previously registered the User The user must have the access token, the ca root certificate and the URLs to onboard an Invoker. Service published in CAPIF Log Entry exist in CAPIF
Execution Steps	1. Onboard Invoker and Provider at CCF. 2. Publish Service API 3. Create Log Entry 4. Get Log Entry

Information of Test	1. Onboard Invoker and provider at CCF.
	2. Publish Service API at CCF:
	  a. Send **POST** to ccf_publish_url https://{CAPIF_HOSTNAME}/published-apis/v1/{apfId}/service-apis
	  b. body [service api description] with apiName **service_1**
	  c. Use **APF Certificate**
	3. Log Entry:
	  a. Send **POST** to https://{CAPIF_HOSTNAME}/api-invocation-logs/v1/{aefId}/logs
	  b. Body **InvocationLog**
	  c. Use **AEF Certificate**
	4. Get Log:
	  a. Send **GET** to https://{CAPIF_HOSTNAME}/logs/v1/apiInvocationLogs?aef-id={aefId}&api-invoker-id={api-invoker-id}
	  b. Use **AMF Certificate**

Expected Result	1. Response to Logging Service must accomplish: a. **200 OK** b. Response Body must follow **InvocationLog** data structure with: • aefId • apiInvokerId • logs

Test Case 2: Get CAPIF Log Entry With no Log entry in CAPIF

Test ID	capif_api_auditing-2
Description	This test case will check that a CAPIF AEF cannot get log entry to Logging Service
Pre-conditions	The Administrator must have previously registered the User The user must have the access token, the ca root certificate and the URLs to onboard an Invoker. Service published in CAPIF
Execution Steps	1. Onboard Invoker and Provider at CCF. 2. Publish Service API 3. Get Log Entry

Information of Test	1. Onboard Invoker and provider at CCF.
	2. Publish Service API at CCF:
	  a. Send **POST** to ccf_publish_url https://{CAPIF_HOSTNAME}/published-apis/v1/{apfId}/service-apis
	  b. body [service api description] with apiName **service_1**
	  c. Use **APF Certificate**
	3. Get Log:
	  a. Send **GET** to https://{CAPIF_HOSTNAME}/logs/v1/apiInvocationLogs?aef-id={aefId}&api-invoker-id={api-invoker-id}
	  b. Use **AMF Certificate**

Expected Result	1. Response to Logging Service must accomplish: a. **404 Not Found** b. Error Response Body must accomplish with **ProblemDetails** data structure with: • status 404 • title with message "Not Found Log Entry in CAPIF". • cause with message "Not Exist Logs with the filters applied".

Test Case 3: Get CAPIF Log Entry without aef-id and api-invoker-id

Test ID	capif_api_auditing-3
Description	This test case will check that a CAPIF AMF cannot get log entry to Logging Service

Pre-conditions	The Administrator must have previously registered the User The user must have the access token, the ca root certificate and the URLs to onboard an Invoker. Service published in CAPIF Log Entry exist in CAPIF
Execution Steps	1. Onboard Invoker and Provider at CCF. 2. Publish Service API 3. Create Log Entry 4. Get Log Entry
Information of Test	1. Onboard Invoker and provider at CCF. 2. Publish Service API at CCF: a. Send **POST** to ccf_publish_url https://{CAPIF_HOSTNAME}/published-apis/v1/{apfId}/service-apis b. body [service api description] with apiName **service_1** c. Use **APF Certificate** 3. Log Entry: a. Send **POST** to https://{CAPIF_HOSTNAME}/api-invocation-logs/v1/{aefId}/logs b. Body **InvocationLog** c. Use **AEF Certificate** 4. Get Log: a. Send **GET** to https://{CAPIF_HOSTNAME}/logs/v1/apiInvocationLogs b. Use **AMF Certificate**
Expected Result	1. Response to Logging Service must accomplish: a. **400 Bad Request** b. Error Response Body must accomplish with **ProblemDetails** data structure with: • status 400 • title with message "Bad Request" • detail with message "aef_id and api_invoker_id parameters are mandatory". • cause with message "Mandatory parameters missing".

Test Case 4: Get CAPIF Log Entry with filter api-version

Test ID	capif_api_auditing-4
Description	This test case will check that a CAPIF AMF can get log entry to Logging Service with filter api-version
Pre-conditions	The Administrator must have previously registered the User The user must have the access token, the ca root certificate and the URLs to onboard an Invoker. Service published in CAPIF Log Entry exist in CAPIF
Execution Steps	1. Onboard Invoker and Provider at CCF. 2. Publish Service API 3. Create Log Entry 4. Get Log Entry
Information of Test	1. Onboard Invoker and provider at CCF. 2. Publish Service API at CCF: a. Send **POST** to ccf_publish_url https://{CAPIF_HOSTNAME}/published-apis/v1/{apfId}/service-apis b. body [service api description] with apiName **service_1** c. Use **APF Certificate** 3. Log Entry: a. Send **POST** to https://{CAPIF_HOSTNAME}/api-invocation-logs/v1/{aefId}/logs b. Body **InvocationLog** c. Use **AEF Certificate** 4. Get Log:

	a. Send **GET** to https://{CAPIF_HOSTNAME}/logs/v1/apiInvocationLogs?aef-id={aefId}&api-invoker-id={api-invoker-id}**&api-version={v1}** b. Use **AMF Certificate**
Expected Result	1. Response to Logging Service must accomplish: a. **200 OK** b. Response Body must follow **InvocationLog** data structure with: • aefId • apiInvokerId • logs

Test Case 5: Get CAPIF Log Entry with filter api-version but not exist in log entry

Test ID	capif_api_auditing-5
Description	This test case will check that a CAPIF AMF cannot get log entry to Logging Service with filter api-version
Pre-conditions	The Administrator must have previously registered the User The user must have the access token, the ca root certificate and the URLs to onboard an Invoker. Service published in CAPIF Log Entry exist in CAPIF
Execution Steps	1. Onboard Invoker and Provider at CCF. 2. Publish Service API 3. Create Log Entry 4. Get Log Entry
Information of Test	1. Onboard Invoker and provider at CCF. 2. Publish Service API at CCF: a. Send **POST** to ccf_publish_url https://{CAPIF_HOSTNAME}/published-apis/v1/{apfId}/service-apis b. body [service api description] with apiName **service_1** c. Use **APF Certificate** 3. Log Entry: a. Send **POST** to https://{CAPIF_HOSTNAME}/api-invocation-logs/v1/{aefId}/logs b. Body **InvocationLog** c. Use **AEF Certificate** 4. Get Log: a. Send **GET** to https://{CAPIF_HOSTNAME}/logs/v1/apiInvocationLogs?aef-id={aefId}&api-invoker-id={api-invoker-id}**&api-version={v58}** b. Use **AMF Certificate**
Expected Result	1. Response to Logging Service must accomplish: a. **404 Not Found** b. Error Response Body must accomplish with **ProblemDetails** data structure with: • status 404 • title with message "Parameters do not match any log entry". • cause with message "No logs found ".

Annex E: Change history

Change history							
Date	**Meeting**	**TDoc**	**CR**	**Rev**	**Cat**	**Subject/Comment**	**New version**
2023-10	SA6#57					TR Initial Version as per S6-233407	0.0.0
2023-10	SA6#57					Implementation of the following pCRs approved by SA6: S6-233386	0.1.0
2023-11	SA6#58					Implementation of the following pCRs approved by SA6: S6-233904, S6-233908, S6-233909, S6-233910.	0.2.0
2024-03	SA6#59					Implementation of the following pCRs approved by SA6: S6-240528, S6-240529, S6-240531, S6-240533, S6-240582.	0.3.0
2024-04	SA6#60					Implementation of the following pCRs approved by SA6: S6-241358, S6-241351.	0.4.0
2024-05	SA6#61					Implementation of the following pCRs approved by SA6: S6-242323, S6-242703, S6-242704, S6-242705.	0.5.0
2024-08	SA6#62					Implementation of the following pCRs approved by SA6: S6-243646, S6-243061, S6-243062, S6-243724, S6-243518, S6-243628, S6-243629, S6-243067, S6-243519, S6-243069, S6-243631, S6-243265, S6-243510, S6-243633.	0.6.0
2024-10	SA6#63					Implementation of the following pCRs approved by SA6: S6-244167, S6-244198, S6-244557, S6-244558, S6-244559, S6-244560, S6-244561.	0.7.0
2024-11	SA6#64					Implementation of the following pCRs approved by SA6: S6-245431, S6-245434, S6-245645, S6-245665.	0.8.0
2024-11	SA6#64					Implementation of the following pCR approved by SA6 on email approval: S6-245722, restructuring prior to initial publication.	0.9.0
2024-12	SA#106	SP-241701				Submitted to SA#106 for information and approval	1.0.0
2024-12	SA#106	SP-241701				MCC Editorial update for publication after TSG SA approval (SA#106)	18.0.0

History

Document history		
V18.0.0	January 2025	Publication