SNAP BI (proposed)

1 Overview

The SNAP API Profile provides a description of the elements that are common across all the SNAP APIs.

This profile should be used in conjunction with compatible functional profiles (such as Accounts and Transactions or Payments) and compatible resources.

1.1 Document Structure

This document consists of the following parts:

- **Overview**: Provides an overview of the profile and the key decisions and principles.
- **Basics**: The section begins with an introduction to how the APIs are used.
- **Security & Access Control**: Specifies the means for TPPs and PSUs to authenticate themselves and provide consent.
- **Normative References**: List of reference.

1.2 Design Principles

a. RESTful APIs

The API adheres to RESTful API concepts where possible and sensible to do so.

However, the priority is to have an API that is simple to understand and easy to use. In instances where following RESTful principles would be convoluted and complex, the principles have not been followed.

References:

- The highest level Data Description Language used is the JSON Schema : http://json-schema.org/
- Best Practice has also been taken from the Data Description Language for APIs; JSON API : http://jsonapi.org/
- The Interface Description Language used is the Swagger Specification version 2.0 (also known as Open API) : http://swagger.io/ (https://github.com/OAI/OpenAPI-Specification)

b. Standards

The SNAP principles for developing API standards:

- SNAP will adopt existing standards where relevant/appropriate to minimize re-inventing the wheel.
- The Standards currently being reviewed include ISO20022 link and FAPI link.
- The adoption currently being reviewed include Open Banking UK link.
- SNAP will favor developer/user experience over and above adoption of existing Standards, in order to create a more future proof Standard.

- Where relevant, the API payloads have been **flattened** so that they are more developer friendly.
- Only elements that are required for the functioning of the API endpoint will be included in the API payload. **API endpoints are defined for specific use-cases** (not to be generically extensible for all use-cases).

c. Extensibility

It is intended that the API flows will be extended to cater for more complex use-cases in subsequent releases, and we have kept this in mind during the design.

d. Idempotency

Idempotency is difficult to implement consistently and leverage consistently.

As a result, idempotency is used sparingly in the SNAP API specifications; with a preference to allow TPPs to simply re-submit a request under failure conditions.

APIs have been defined to be idempotent, where not doing so would cause a poor PSU user-experience or increase false positive risk indicators.

e. Message Signing

Digital signatures will facilitate non-repudiation for SNAP APIs.

The approach for message signing is documented in Basics / Message Signing.

The applicability of signatures to individual requests and responses is documented on the page for each of the resources. However, implementers of the standards can **optionally** add signatures to all response and request payloads.

f. Message Encryption

Message Encryption is on **rod-map** feature of the SNAP APIs to facilitate additional protection of inflight data.

g. Agnostic to Payment Schemes

The API will be designed so that it is agnostic to the underlying payment scheme that is responsible for carrying out the payment. As a result, we will not design field lengths and payloads to only match the Faster Payments message but will consider other payment variants.

h. Status Codes

The API uses two status codes that serve two different purposes:

- The HTTP Status Code reflects the outcome of the API call (the HTTP operation on the resource). Granular Functional Error Codes are specified as part of API Error Response Structure.
- A Status field in some of the resource payloads reflects the status of resources.

i. Unique Identifiers (Id Fields)

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

However, considering that some of the resources described in these specifications do not have a primary key in the system of record, the Id field will be optional for some resources.

An ASPSP that chooses to populate optional Id fields must ensure that the values are unique and immutable.

j. Categorization of Implementation Requirements

Where a requirement is being implemented by either an ASPSP and/or a TPP, a different categorization is applied. The functionality, endpoints and fields within each resource are categorized as 'Mandatory', 'Conditional' or 'Optional'.

ASPSPs **must** make documentation available to TPPs (e.g., on their developer portals) to which 'Conditional' / 'Optional' endpoints and fields are implemented for any given implementation of the specification.

j.1 Mandatory

Functionality, endpoints, and fields marked as Mandatory are required in all cases for regulatory compliance and/or for the API to function and deliver essential customer outcomes.

For functionalities and endpoints:

- An ASPSP **must** implement an endpoint that is marked Mandatory.
- An ASPSP **must** implement functionality that is marked Mandatory.

For fields:

- A TPP **must** specify the value of a Mandatory field.
- An ASPSP **must** process a Mandatory field when provided by the TPP in an API request.
- An ASPSP **must** include meaningful values for Mandatory fields in an API response.

j.2 Conditional

Functionality, endpoints, and fields marked as Conditional may be required in some cases for regulatory compliance (for example, if these are made available to the PSU in the ASPSP's existing Online Channel, or if ASPSPs (or a subset of ASPSPs) have been mandated by a regulatory requirement).

For functionalities and endpoints:

- An ASPSP **must** implement functionality and endpoints marked as Conditional if these are required for regulatory compliance.

For fields:

- All fields that are not marked as Mandatory are treated as Conditional.
- A TPP **may** specify the value of a Conditional field.
- An ASPSP **should** process a Conditional field when provided by the TPP in an API request and **should** respond with an error if it cannot support a particular value of a Conditional field.
- An ASPSP **should** include meaningful values for Conditional fields in an API response if these are required for regulatory compliance.

j.3 Optional

Functionality and endpoints marked as Optional are not necessarily required for regulatory compliance but may be implemented to enable desired customer outcomes.

For functionalities and endpoints:

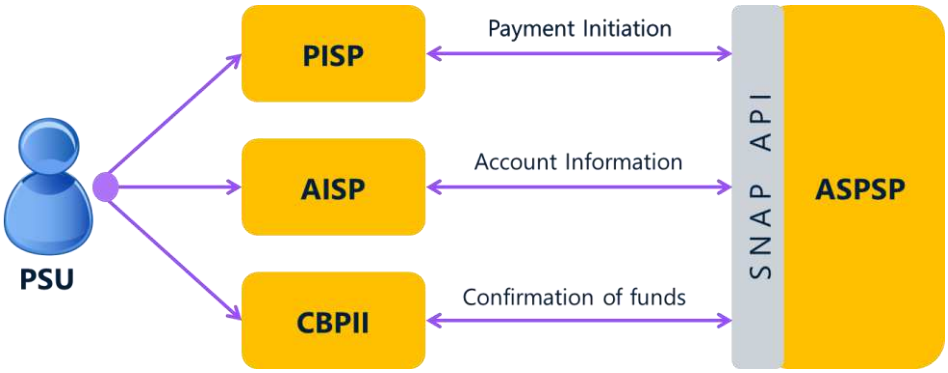
- An ASPSP **may** implement an Optional endpoint.
- An ASPSP **may** implement Optional functionality.

For fields:

- All fields that are not marked as Mandatory are treated as Conditional.
- For any endpoints which are implemented by an ASPSP, the fields are either Mandatory or Conditional.

2 Basics

2.1 Outline



General outline

2.2 Actors

Actor	Abbreviation	Type	Specializes	Description
Payment Service User	PSU	Person	N/A	A natural or legal person making use of a payment service as a payee, payer, or both.
Payment Service Provider	PSP	Legal Entity	N/A	A legal entity (and some natural persons) that provide payment services.
Account Servicing Payment Service Provider	ASPSP	Legal Entity	PSP	An ASPSP is a PSP that provides and maintains a payment account for a payment services user.
Third Party Providers / Trusted Third Parties	TPP	Legal Entity	PSP	A party other than an ASPSP that provides payment related services. The term is generally deemed to include all payment service providers that are 3rd parties (the ASPSP and the PSU to whom the account belongs being the first two parties). References to a "TPP" in the specification relate to a piece of registered software with an ASPSP (with a specific client_id).
Payment Initiation Service Provider	PISP	Legal Entity	TPP	A TPP that provides Payment Initiation Services. A PSP that provides Payment Initiation Services.

Actor	Abbreviation	Type	Specializes	Description
Account Information Service Provider	AISP	Legal Entity	TPP	A TPP that provides Account Information Services. A PSP that provides account information services
Card Based Payment Instrument Issuer	CBPII	Legal Entity	TPP	A TPP that issues electronic or card-based payment instruments to PSUs and requires access to the Confirmation of Funds API.

2.3 Character Encoding

The API requests and responses **must** use a UTF-8 character encoding. This is the default character encoding for JSON (RFC 7158 - Section 8.1)

However, an ASPSP's downstream system may not accept some UTF-8 characters, such as emoji characters (e.g. "Happy Birthday 🤯🤯!" may not be an acceptable Payment Reference). If the ASPSP rejects the message with a UTF-8 character that cannot be processed, the ASPSP **must** respond with an HTTP 400 (Bad Request) status code.

2.4 Date Formats

All dates in the JSON payloads are represented in ISO-8601 date-time format. All date-time fields in responses **must** include the timezone. For Example:

```
2022-04-05T10:43:07+0700
2021-07-03T14:43:41Z
```

All dates in the query string are represented in ISO-8601 date-time format and **must not** include the timezone. For example:

```
2017-04-05T10:43:07
2017-04-05
```

All dates in the HTTP headers are represented as RFC 7231 Full Dates. For example:

```
Sun, 10 Sep 2022 19:43:31 GMT
```

All dates in the JWT claims are expressed as a JSON number, representing the number of seconds from 1970-01-01T0:0:0Z as measured in GMT until the date/time.

```
//Sun, 12 Feb 2018 14:45:00 GMT
1518446700
```

When an ASPSP receives a request with an incorrectly formatted date, it **may** respond with a status code of 400 Bad Request and an error code.

2.5 Resource URI Path Structure

The path of the URI must follow the structure below.

- **[participant-path-prefix]/snap/{snap-version}/ [api-category]/[api-category-version]/[resource]/[resource-id]/[[sub-resource]]**

This consists of the following elements:

- **[participant-path-prefix]**
  An optional ASPSP specific path prefix.
- **snap**
  The constant string "snap".
- **[snap-version]**
  The version of the SNAP APIs expressed as /[major-version].[minor-version]/.
- **[api-category]**
  The API category identifies the group of endpoints.
- **[api-category-version]**
  The version of the API category expressed as /[major-version].[minor-version]/.
- **[resource]/[resource-id]**
  Details the resource.
- **[[sub-resource]]**
  Details the sub-resources expressed as /[sub-resource] [0..2]/.

ASPSP **must** use the same [participant-path-prefix] and host name for all its resources. Examples:

- https://superbank.com/apis/snap/1.0/card/1.0/card
- https://api.bank.com/snap/1.0/card/1.1/card/ty56w1-772635-2837-8
- https://api.bank.com/snap/1.0/card/1.1/card/ty56w1-772635-2837-8/lifecycle/bind

For brevity, the APIs are referred to by their resource names in these documents and in all examples.

2.6 Headers

a. Request Headers

Header Parameters	Notes	Request Method			
		POST	GET	DELETE	PUT
x-auth-date	The time when the CON last logged in with the TPP. The value is supplied as a HTTP-date as in section 7.1.1.1 of [RFC7231], e.g., `x-auth-date: Tue, 11 Sep 2012 19:43:31 GMT`	O	O	O	Do not use
x-customer-ip-address	The PSU's IP address if the PSU is currently logged in with the TPP. As PSU present indicator. ex: in X-Forwarded-For HTTP header.	O	O	O	Do not use
x-interaction-id	An RFC4122 UID used as a correlation Id. If provided, the ASPSP **must** "play back" this value in the x-interaction-id response header.	O	O	O	O
x-api-key	An unique identifier that authenticates requests associated with client application for usage and billing purposes.	M	M	M	M
Authorization	Standard HTTP Header. Allows Credentials to be provided to the Authorization / Resource Server depending on the type of resource being requested. For OAuth 2.0 / OIDC, this comprises of either the Basic / Bearer Authentication Schemes.	M	M	M	M
Content-Type	Standard HTTP Header; Represents the format of the payload being provided in the request. This **must** be set to `application/json`, except for the endpoints that support Content-Type other than `application/json` (e.g POST /file-payment/{Id}/file), the ASPSP must specify the available options on their developer portals. This **must** be set to `application/jose+jwe` for encrypted requests. If set to any other value, the ASPSP **must** respond with a 415 Unsupported Media Type.	M	Do not use	Do not use	M
Accept	Standard HTTP Header; Determine the Content-Type that is required from the Server. For endpoints that do not respond with JSON (e.g GET ../statements/{StatementId}/file), the ASPSP must specify the available options on their developer portals. If set to an unacceptable value the ASPSP **must** respond	O	O	Do not use	O

Header Parameters	Notes	Request Method			
		POST	GET	DELETE	PUT
	with a 406 (Not Acceptable). If not specified, the default is application/json.				
x-idempotency-key	Custom HTTP Header; Unique request identifier to support idempotency. Mandatory for POST requests to idempotent resource end-points. Must not be specified for other requests.	O	Do not use	Do not use	Do not use
x-jws-signature	Header containing a detached JWS signature of the body of the payload. Refer to resource specific documentation on when this header must be specified.	API specific	API specific	API specific	M
x-customer-user-agent	The header indicates the user-agent that the PSU is using. The TPP **may** populate this field with the user-agent indicated by the PSU. If the PSU is using a TPP mobile app, the TPP **must** ensure that the user-agent string is different from browser based user-agent strings.	O	O	O	O
x-customer-device-token	A globally unique ID identifying the user's client device or user's user-agent. The TPP **may** populate this field indicated by the PSU. Device token is property that allows additional context for the authentication, or device behavior detection.	O	O	O	O
x-customer-geo-location	The forwarded Geo Location of the corresponding HTTP request between PSU and TPP if available. Format value `GEO:<latitude>;<longitude>` e.g., GEO:52.506931;13.144558	O	O	O	O

Whether the PSU is present or not-present is identified via the x-customer-ip-address header. If the PSU IP address is supplied, it is inferred that the PSU is present during the interaction.

The implications to this are:

- ASPSPs will need to rely on the AISPs assertion.

b. Response Headers

Header Parameters	Notes	Mandatory?
Content-Type	Standard HTTP Header; Represents the format of the payload returned in the response. The ASPSP **must** return `Content-Type: application/json` as a content header for all unencrypted endpoints, except the GET ../statements/{StatementId}/file and ../file-payment/{Id}/file endpoints, where it is up to the ASPSP to specify available options. The ASPSP **must** return `Content-type: application/jwe` for all encrypted end-points.	Mandatory
x-jws-signature	Header containing a detached JWS signature of the body of the payload. Refer to resource specific documentation on when this header **must** be returned. Where a signed response is indicated in the documentation this header **should** be returned for error responses where a response body is returned.	API specific
x-interaction-id	An RFC4122 UID used as a correlation Id. The ASPSP **must** set the response header `x-interaction-id` to the value received from the corresponding client request header or to a RFC4122 UUID value if the request header was not provided to track the interaction. The header **must** be returned for both successful and error responses.	Mandatory
Retry-After	Header indicating the time (in seconds) that the TPP should wait before retrying an operation. The ASPSP **should** include this header along with responses with the HTTP status code of 429 (Too Many Requests).	Optional

2.7 Common Payload Structure

Response Structure

Two additional top-level sections are included in the response for:

- _links
- _meta

b.1 _links

The _links section is mandatory and will always contain absolute URIs to related resources,

The "self" member is mandatory.

For example:

```
"_links": {
   "self": "../snap/1.0/card/1.1/card/ty56w1-772635-23",
   "activate": "../snap/1.0/card/1.1/card/ty56w1-772635-23/bind",
   ...
   "logo": "http://example.com/img/logo.png"
}
// _links: ["self", "activate", "authorized", "cancel", "verify", "deactivate", "suspend", "unsuspend",
// "approve", "deny", "expire", "status", "prev", "next", "start", "end", "logo"]
```

b.2 _meta

The _meta section is optional. An optional member is "totalPages" which shows how many pages of results (for pagination) are available.
For example:

```
"_meta": {
   "totalPages": "https://api.bank.com/snap/1.0/card/1.1/card/ty56w1-772635-23",
   "firstAvailableDateTime": "2022-05-28T00:00:00+00:00",
   "lastAvailableDateTime": "2022-12-06T00:00:00+00:00"
}
```

b.3 Error Response Structure

The error response structure for SNAP APIs:

```
{
  "Error": {
     "code": ". . .",
            //1..1, High level textual error code, to help categorize the errors.
               //ex: Header.Invalid.01, Resource.ConsentMismatch, UnexpectedError tbd
     "id": ". . .",
            //0..1, A unique reference for the error instance,
               //for audit purposes, in case of unknown/unclassified errors.
     "message": ". . .",
            //1..1, A description of the error that occurred.
               //e.g., 'Unable to retrieve balance info'
     "systrace": ". . ."
            //0..1, (char256) the low level error trace.
               //e.g., 'Error in http connection java.net.UnknownHostException.'
  }
}
```

b.4 Optional Fields

In objects where the value for an optional field is not specified, the field **must** be excluded or set null from the JSON payload.

In objects where an array field is defined as having 0..n values (zero length or recurrence empty), the array field **must be** included in the payload with an empty array. E.g.:

```
{
    "name": "",       // Incorrect. Exclude/null the name field from the payload.
    "account": {},    // Incorrect. Exclude/null the creditor account field.
    "factors": [{"type": ""}],    // incorrect.
    "balances": []    // Correct. This is the method of indicating an empty array.
                      //Do not suppress the balances field.
}
```

2.8 HTTP Status Codes

The following are the HTTP response codes for the different HTTP methods, across all SNAP API endpoints.

Situation	HTTP Status	Notes	Returned by			
			POST	GET	DELETE	PUT
Request completed successfully	200 OK	PUT will be specified to return the updated resource. A 200 status code is therefore appropriate.	Yes	Yes	No	Yes
Normal execution. The request has succeeded.	201 Created	The operation results in the creation of a new resource.	Yes	No	No	No
Delete operation completed successfully	204 No Content		No	No	Yes	No
Request has malformed, missing or non-compliant JSON body, URL parameters or header fields.	400 Bad Request	The requested operation will not be carried out.	Yes	Yes	Yes	Yes
Authorization header missing or invalid token	401 Unauthorized	The operation was refused access. Re-authenticating the PSU may result in an appropriate token that may be used.	Yes	Yes	Yes	Yes
Token has incorrect scope, or a security policy was violated.	401 Unauthorized	The operation was refused access. Re-authenticating may result in an appropriate token that may be used.	Yes	Yes	Yes	Yes

Situation	HTTP Status	Notes	Returned by			
			POST	GET	DELETE	PUT
Request accessing a resource that it does not have permission to access.	403 Forbidden	The operation was refused to proceed.	Yes	Yes	Yes	Yes
The TPP tried to access the resource with a method that is not supported.	405 Method Not Allowed		Yes	Yes	Yes	Yes
The request contained an Accept header other than permitted media types and a character set other than UTF-8	406 Not Acceptable		Yes	Yes	Yes	Yes
The operation was refused because the payload is in a format not supported by this method on the target resource.	415 Unsupported Media Type		Yes	No	No	Yes
The operation was refused as too many requests have been made within a certain timeframe.	429 Too Many Requests	ASPSPs **may** throttle requests when they are made in excess of their fair usage policy. ASPSPs **must** document their fair usage policies in their developer portals. The ASPSP **must** respond with this status if it throttles the request. The ASPSP **should** include a Retry-After header in the response indicating how long the TPP must wait before retrying the operation.	Yes	Yes	Yes	Yes
Something went wrong on the API gateway or micro-service	500 Internal Server Error	The operation failed.	Yes	Yes	Yes	Yes

Situation	HTTP Status	Notes	Returned by			
			POST	GET	DELETE	PUT
Service version deprecation	503 Service Unavailable	Where an API is deprecated and no longer operationally supported by an ASPSP, its URI path may still be active and accept API requests. In this context it is recommended that a 503 Service Unavailable be returned so that the TPP is aware of the API version being offline.	Yes	Yes	Yes	Yes

An ASPSP **MAY** return other standard HTTP status codes (e.g., from gateways and other edge devices) as described in RFC 7231 - Section 6.

ASPSPs **must** respond with SNAP Error Response Structure for all errors during API Calls.

a. 400 (Bad Request) v/s 404 (Not Found)

When a TPP tries to request a resource URL with a resource Id that does not exist, the ASPSP **must** respond with a 400 (Bad Request) rather than a 404 (Not Found).

e.g., if a TPP tries to GET /domestic-payments/22289 where 22289 is not a valid DomesticPaymentId, the ASPSP must respond with a 400.

When a TPP tries to request a resource URL that results in no business data being returned (e.g. a request to retrieve standing order on an account that does not have standing orders) the ASPSP **must** respond with a 200 (OK) and set the array to be empty.

If the TPP tries to access a URL for a resource that is not defined by these specifications (e.g. GET /card-accounts), the ASPSP **may** choose to respond with a 404 (Not Found).

If an ASPSP has not implemented an API endpoint, it **must** respond with a 404 (Not Found) for requests to that URL.

The table below illustrates some examples of expected behavior:

Situation	Request	Response
A TPP attempts to retrieve a payment with a DomesticPaymentId that does not exist	GET /domestic-payments/1001	400 (Bad Request)
A TPP attempts to retrieve a resource that is not defined	GET /bulk	404 (Not Found)
A TPP attempts to retrieve a resource that is in the specification, but not implemented by the ASPSP. e.g.,	GET /domestic-scheduled-payments/1002	404 (Not Found)

Situation	Request	Response
an ASPSP has chosen not to implement the status API endpoint for domestic-scheduled-payments		

A TPP attempts to retrieve standing orders for an AccountId that exists, but does not have any standing orders	GET /accounts/1000/transactions	200 OK
		{
		"transactions":[],
		"_links": {
		"self": "../accounts/5678/transactions/"}
		}

b. 403 (Forbidden)

When a TPP tries to access a resource that it does not have permission to access, the ASPSP **must** return a 403 (Forbidden) with an optional body with Error Response.

The situation could arise when:

- The TPP attempted to access a resource with an Id that it does not have access to. e.g., an attempt to access GET /domestic-payments/1001 where a payment resource with Id 1001 belongs to another TPP.
- The TPP tries to access an account/transaction resource and the TPP does not have a consent authorisation with the right Permissions to access the requested resource. e.g., an attempt to access GET /standing-orders when the ReadStandingOrdersBasic permission was not included in the consent authorisation.
- The TPP tries to access an account/transaction resource and the TPP does not have a consent authorisation for the AccountId. e.g., an attempt to access GET /accounts/2001 or GET /accounts/2001/transactions when the PSU has not selected AccountId 2001 for authorization.
- The TPP attempts to access a Resource and the ASPSP decides to re-authenticate the PSU. The ASPSP must respond back with an appropriate error code to indicate re-authentication is required.

c. 401 (Unauthorized)

When a TPP tries to access a resource that it does not have appropriate access token neither the scope to access, the ASPSP **must** return a 401 (Unauthorized) with an optional body with Error Response.

When the TPP uses an expired access token, the ASPSP **must** return a 401 (Unauthorized) with an optional body with Error Response.

The situation could arise when an ASPSP has chosen to expire an Access Token. Here are few reasons:

1. The consent has expired (the Expiration Date Time has lapsed)
2. Suspicious usage of the Access Token or suspected fraud
3. SCA is required in permitted circumstances

This error can potentially be remedied by asking the PSU to re-authenticate or authenticate with the right permissions.

d. 429 (Too Many Requests)

When a TPP tries to access a resource too frequently the ASPSP **may** return a 429 (Too Many Requests). This is a non functional requirement and is down to individual ASPSPs to decide throttling limits.

This situation could arise when:

- A TPP decides to implement "Real Time Payment Status" functionality for its users and implements this badly by polling a GET endpoint or an Idempotent POST endpoint in excess of the ASPSP's fair usage policy to provide pseudo "real-time" Status updates to the user.
- A TPP decides to use the Single Immediate Payment endpoint as if it were a batch payment facility and sends a large number of payment requests in a very short space of time such that it exceeds the ASPSP's fair usage policy.

2.9 Idempotency

An idempotency key is used to guard against the creation of duplicate resources when using the **POST** API endpoints (where indicated).

If an idempotency key is required for an API endpoint:

- The x-idempotency-key provided in the header **must** be at most 40 characters in size. If a larger x-idempotency-key length is provided, the ASPSP **must** reject the request with a status code is 400 (Bad Request).
- The TPP **must not** change the request body while using the same x-idempotency-key. If the TPP changes the request body, the ASPSP **must not** modify the end resource. The ASPSP **may** treat this as a fraudulent action.
- The ASPSP **must** treat a request as idempotent if it had received the first request with the same x-idempotency-key from the same TPP in the **preceding 24 hours**.
- The ASPSP **must not** create a new resource for a POST request if it is determined to be an idempotent request.
- The ASPSP **must** respond to the request with the current status of the resource (or a status which is at least as current as what is available on existing online channels) and a HTTP status code of 201 (Created).
- The TPP **must not** use the idempotent behaviour to poll the status of resources.
- The ASPSP **may** use the message signature, along with the x-idempotency-key to ensure that the request body has not changed.

If an idempotency key is not required for an API endpoint:

- The ASPSP **must** ignore the idempotency key if provided.

2.10 Message Signing

a. Overview

This section provides an overview of how message signing is implemented for the SNAP APIs.

The APIs require TLS 1.2/1.3 Mutual Authentication, and this may be used as a means of non-repudiation. However, it would be difficult to maintain digital records and evidence of non-repudiation if the API only relied on TLS 1.2/1.3.

A solution for non-repudiation that does not rely on TLS, would be achieved by providing a JWS with detached content (as defined in RFC 7515 - Appendix F) in the HTTP header of each API request.

```
x-jws-signature:
V2hhdCBoYXRoIGdvZCB3cm91Z2h0ID8=..QnkgR2VvcmdlLCBzaGUncyBnb3QgaXQhIEJ5IEdlb3JnZSBzaGUn
cyBnb3QgaXQhIE5vdyBvbmNlIGFnYWluLCB3aGVyZSBkb2VzIGl0IHJhaW4/
```

The HTTP body would form an un-encoded payload as defined in RFC 7797. In addition, the un-encoded payload has: no space, no tab, no newline, and no line break.

The JWS **must** be signed using an algorithm that supports asymmetric keys (*public-private certificate* pairs).

A request would be signed by a TPP's private key, and a response would be signed by the ASPSP's private key.

Not all API requests and responses are signed. Whether message signing is mandatory, supported or not supported is documented along with each API and per ASPSP specific.

b. Specification

The TPP **must** sign the HTTP body of each API request that requires message signing.

The ASPSP **must** sign the HTTP body of each API response that requires message signing.

The ASPSP **should** verify the signature of API requests that it receives before carrying out the request. If the signature fails validation, the ASPSP **must** respond with a 400 (Bad Request).

The ASPSP **must** reject any API requests that should be signed but do not contain a signature in the HTTP header with a 400 (Bad Request) error.

The TPP **should** verify the signature of API responses that it receives. If the signature fails validation, the TPP **must** ignore/cancel/stop/reverse preceding intend.

The signer **must** sign the message with either ES256, PS256 or RS256.

3 Security & Access Control

3.1 Technical Characteristics and Security Baseline

1. HTTP v1.1
2. mTLS 1.2 or higher
3. REST/JSON; charset UTF-8
4. Data model origin: ISO 20022
5. X.509 Certificate
6. OpenID Connect + PKCE
7. OAuth2 Authorization Framework
8. JWS (JSON Web Signature)
9. Strong Customer Authentication (SCA)
10. Consent Authorization:
    - an access token is bound to a single PSU (user) and an intent.
11. Object/resource-Entity Authorization:
    - shall identify the associated entity to the access token
    - shall only return the resource identified by the combination of the entity implicit in the access and the granted scope
12. Data exchange confidentiality
    - all parties **must** limit their trust as much as possible to avoid possible exchange of sensitive data with possible adversaries
13. User intent re-Authentication (per API specific)
14. Idempotency (per API specific)

3.2 Scopes & Grant Types

To access each of the APIs, the API must be called with an access token in the HTTP `Authorization` header. The scopes required with these access tokens and the grant type used to get the access token are specified in the specific API documentation.

3.3 Resource owner Authorization

The APIs require access token (as proof-of-possession bearer tokens) to be provided to the Authorization / Resource Server depending on the type of resource being requested.
The type of resource being requested is categorized by the type of the resource owner (user) being identified. Type of resource owner being identified are:

1. User present
2. User not present
3. User mandate (user present, deprecated)

The standard assumes that the PSU may one set of identifiers to identify itself to the ASPSP and another distinct set of identifiers to identify itself to the TPP. The standard does not assume that the ASPSP and TPP must share the PSU's password as this is inherently a poor security practice.

Type of Authorization flow mechanism as per resource owner being identified:

Resource owner identified	APIs category	Authorization
User present	B2C / B2B2C	OpenID Connect `authorization`
User not present	B2B	OAuth2 `client_credentials` JWT
User mandate (present, no user consent) (deprecated)	B2B2C	OAuth2 *custom* `authorization`

Whether the PSU is present or not-present is identified via the x-customer-ip-address header.

3.4 Consent Authorization

OAuth2.0 scopes are coarse grained, and the set of available scopes are defined at the point of client registration. There is no standard method for specifying and enforcing fine grained scopes (e.g., a scope to enforce payments of a specified amount on a specified date).

An *intent* is used to define the fine-grained permissions that are granted by the PSU to the TPP.

The act of providing authorization of an intent by a PSU to an ASPSP is called *consent authorization*.

The SNAP APIs use a variety of intents such as account-access-consent, funds-confirmation-consents, and the payment-order-consents.

A TPP requests an ASPSP to create intent by using a client credentials grant. The ASPSP creates the intent and responds with the intent-id.

The TPP then redirects the PSU to the ASPSP to authorize consent for the intent, passing in an intent-id as a parameter.

This is done through an authorization grant flow and results in the issuance of an access token tied to the authorized intent.

An access token is bound to a single PSU and an intent.

A call to GET /**{apis}-consent/{ConsentId}** will always be the first call once a valid access token granted, this allow ASPSP updating the consent status from "AwaitingAuthorisation" to be "Authorised".
A TPP can requests an ASPSP to get created resources information by using a client credentials grant.

3.5 Consent Authorization Status code-list

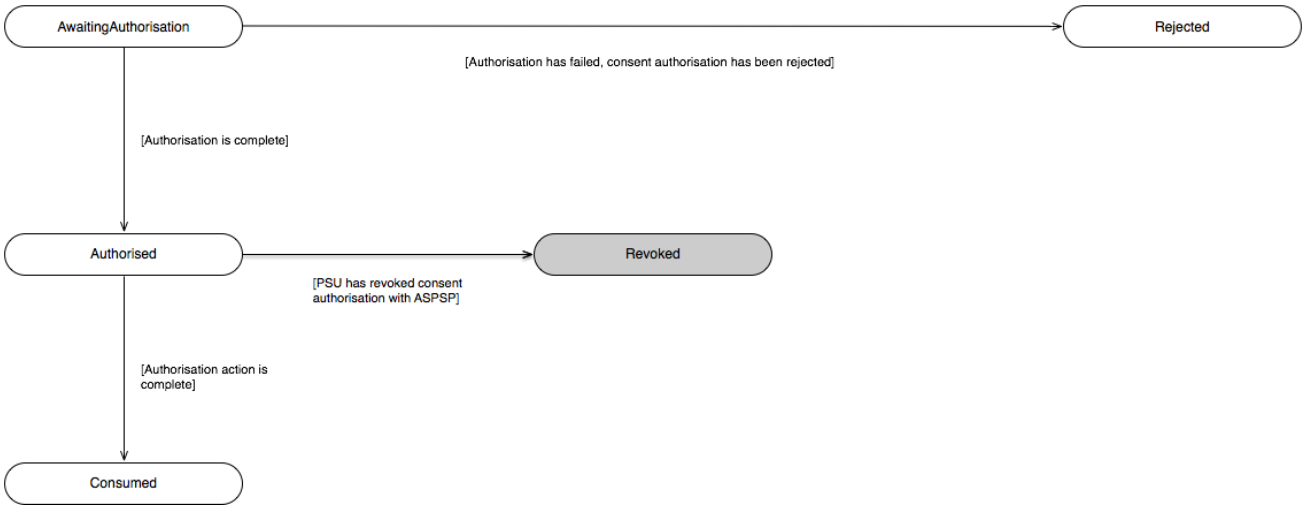
The consent resource that is created successfully must have the following Status code-list enumeration:

	Status	Status Description
1	AwaitingAuthorisation	The consent resource is awaiting authorisation.

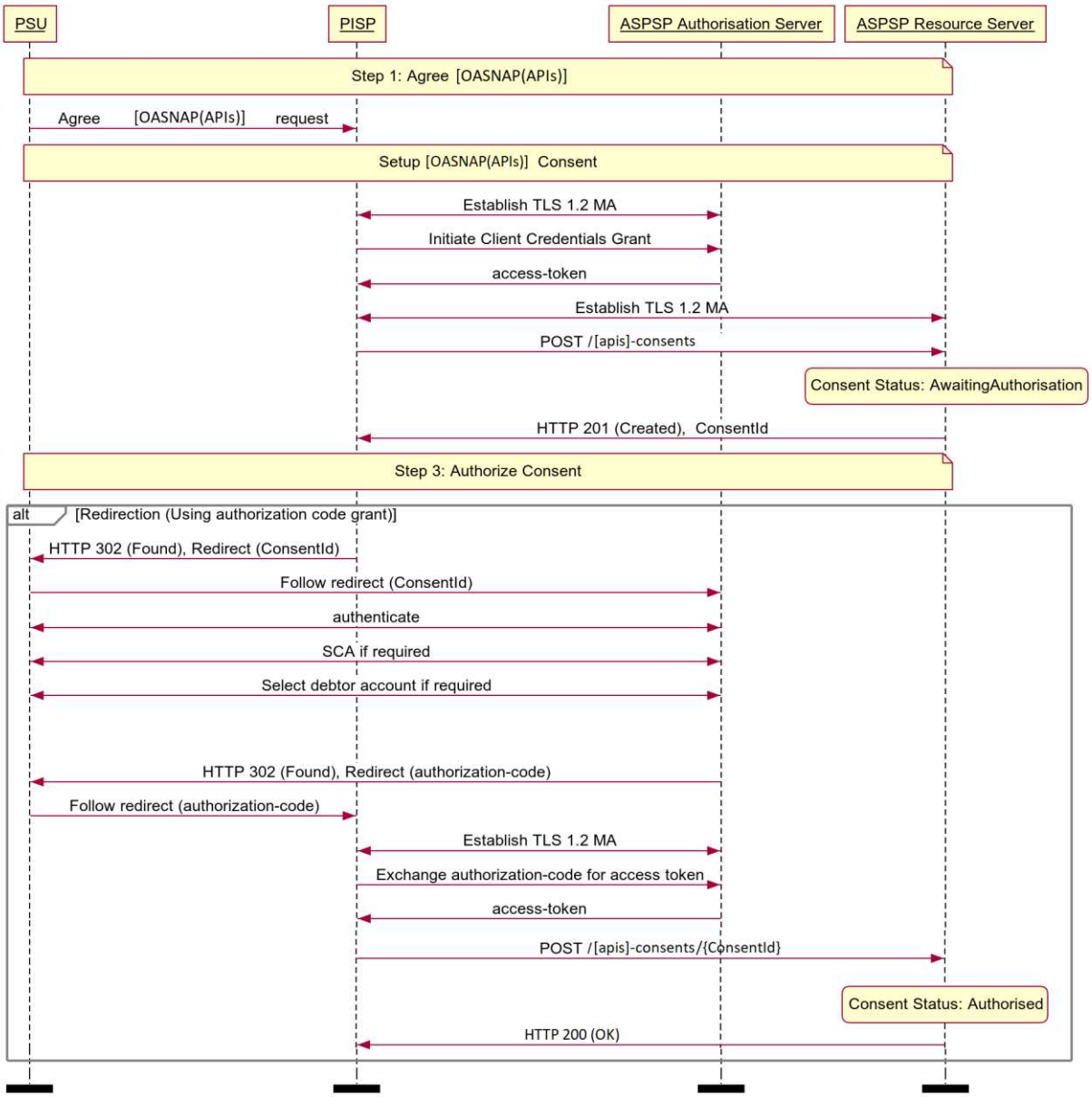
After authorisation has taken place the consent resource may have these following statuses.

	Status	Status Description
1	Rejected	The consent resource has been rejected.
2	Authorised	The consent resource has been successfully authorised.
3	Consumed	The consented action has been successfully completed. This does not reflect the status of the consented action. Specifically for Payment consent related.
4	Revoked	The consent resource has been revoked via the ASPSP interface.

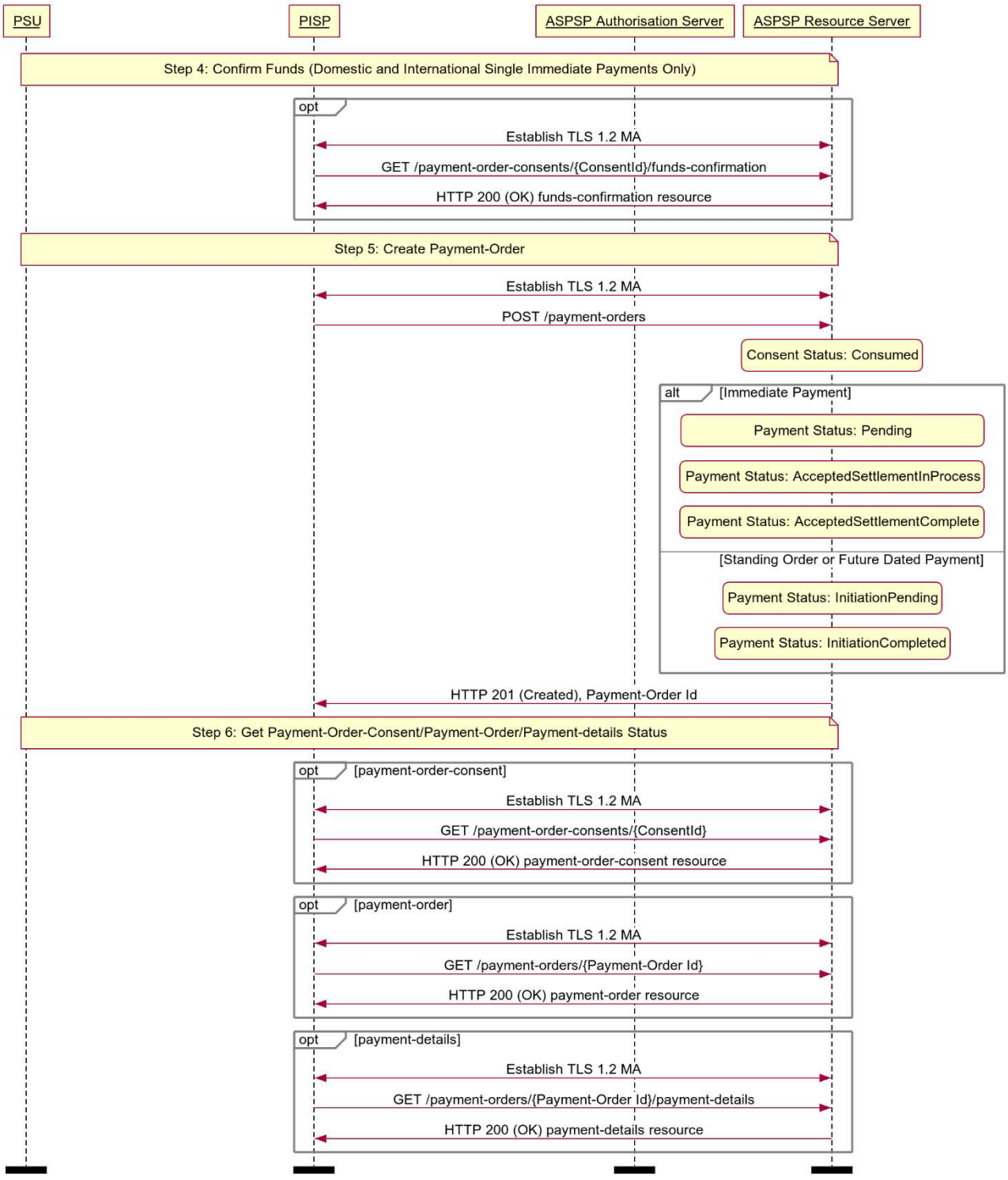
Consent state model:



3.6 Consent Authorization Framework



3.7 Payment Consent Authorization Framework



3.8 User intent re-Authentication

Once the payment-consent has been authorised by the PSU, the PISP can proceed to submitting the **/payment** for processing by making a POST request to the payments endpoint.

To re-ensure the intent of the user action, ASPSP may require additional SCA identification when the consent status already "Consumed".

This intent for user re-authentication is addressed by additional header named: x-factor.

Header Parameters	Notes	Request Method			
		POST	GET	DELETE	PUT
x-factor	One time password or Time-based One Time Password for payment intent re-authentication. The value may be delivered in forms of decouple/push notification to ASPSP *authorization device*.	O	Do not use	Do not use	Do not use

3.9 Length of Authorization Code, Access Token and Refresh Token

- In accordance with the OAuth 2.0 Framework, TPPs **must** expect that the length of the authorization code, access token and refresh token, where applicable, may change over time as ASPSPs make changes to what is stored in them and how they are encoded. TPPs **should** expect that they will grow and shrink over time.
- TPPs **must** use a variable length data type without a specific maximum size to store these attributes.
- An ASPSP, prior to any increase in the length of Authorization Code, Access Token and Refresh Tokens, **must** consider the limits enforced by the common tools available in the market to publish and consume Authorization Code, Access Token and Refresh Tokens. Violation of these limits may break the TPPs, using those browsers/tools.

3.10 Use of Refresh Token

An ASPSP may issue a refresh token along with an access token as a result of consent re-authentication.

When an access token expires, the TPP may attempt to get a new access and refresh token as defined in Section 6 of the OAuth 2.0 specification.

4 Normative References

4.1 Open Banking Reference

1. Open Banking UK: https://www.openbanking.org.uk/
2. ISO 20022: https://www.iso20022.org/iso-20022-message-definitions

4.2 Common Reference

1. Hypertext Transfer Protocol (HTTP/1.1): RFC 2616 and RFC 7231
2. JSON Schema : http://json-schema.org/
3. JSON API : http://jsonapi.org/
4. UTF-8 character encoding: RFC 7158 - Section 8.1
5. A Universally Unique IDentifier (UUID): RFC 4122
6. Interface Description Language used is
   the Swagger Specification v2.0 (also known as Open API): http://swagger.io/
   (https://github.com/OAI/OpenAPI-Specification)

4.3 Security Profile Reference

1. Financial-grade API Security Profile 1.0 - Part 1: Baseline: https://openid.net/wg/fapi/
2. Financial-grade API Security Profile 1.0 - Part 2: Advanced: https://openid.net/wg/fapi/
3. The OAuth 2.0 Authorization Framework: RFC 6749
4. OAuth 2.0 Bearer Token: RFC 6750
5. OpenID Connect: https://openid.net/connect/
6. JSON Web Token (JWT): RFC 7519
7. JSON Web Signature (JWS): RFC 7515 and RFC 7797
8. JSON Web Algorithms (JWA): RFC 7518