Pensions Dashboards Programme

Technical standards

January 2022

This scoping document describes the baseline design of APIs required to interact with the pensions dashboards ecosystem.

pensionsdashboardsprogramme.org.uk

Money & Pensions Service

1 Introduction

1.1 Purpose

This document outlines the API baseline design for PDP specific APIs, as part of our technical standards.

1.2 Scope

The following areas will be covered in the document:

- *a summary of each API*
- *hosting*
- *format*
- *HTTP method*
- *authorisation*
- *expected responses*
- *error handling*

1.3 Transaction monitoring

All interfaces will carry a unique transaction identifier for logging, audit and monitoring purposes, the detail of which will be published in future documentation.

2 Find API

2.1 Summary of the find API

The find API is exposed by data providers and its purpose is to receive the find requests, which are initiated by the pension finder service (PFS) containing the pension owner's personally identifiable information (PII) data, needed for a data provider to use in order to determine a match within the internal records. The find input data will also carry the relevant consents given by the pension owner during the find process and the user account token needed for obtaining the protected API token (PAT) needed for the data providers to use the UMA specific protection API, to register PeIs upon a successful find, with the consent and authorisation service.

2.2 Pension finder service (PFS)

The pension finder service is orchestration middleware. It distributes the find request across the data provider endpoints by invoking their find APIs and manages the low-level interactions to achieve message delivery to the data providers.

2.3 Hosting

Each data provider connected to the ecosystem will be required to host their find API within their domain.

2.4 Format

The find API will be a REST API using JSON encoded as UTF-8.

2.5 Authorisation

This is a closed ecosystem, with all end point connections secured using private public key infrastructure (PKI) certificates issued by the Pensions Dashboards Programme (PDP) governance register to suitably enrolled organisations. This enables connecting entities to establish a mutual transport layer security (TLS) connection with the central infrastructure. For find requests only, the PFS will be responsible for invoking the find API endpoints. There is an assumption made by the Pensions Dashboards Programme that there are no additional API security requirements for the find API – this needs to be validated with the industry.

2.6 HTTP method

The PFS will be restricted to only making HTTP POST requests to each data provider find endpoint, with the body of the request containing the find parameters as signed JSON web tokens (JWTs).

Summary of the find request data parameters sent to the data provider find endpoint:

user_token	A combination of the following expressed as a JWT: • *verified identity details such as name, date of birth and postcode. These will have been selected by the consent and authorisation (C&A) service from the verified details supplied by the identity service* • *identity details asserted by the user at the C&A's consent user interface eg, National Insurance Number*
user_account_token	The user account token is an OAuth2 authorisation grant, expressed as a JWT, which can be exchanged for the PAT.
consents_token	A set of user consents for subsequent processing, using the supplied identity details expressed as a JWT.

For example, the PFS makes the following HTTP POST request using mTLS:

POST /find HTTP/1.1
Host: www.dashboard.aviva.com
Content-Type: application/json;charset=UTF-8
{

"user_token":"eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3OD
kwIiwibmFtZSI6IkpvaG4gRG9lIiwiaWF0IjoxNTE2MjM5MDIyfQ.SflKxwRJSMeKKF2Q
T4fwpMeJf36POk6yJV_adQssw5c",
"user_account_token":"eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM
0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiaWF0IjoxNTE2MjM5MDIyfQ.SflKxwRJS
MeKKF2QT4fwpMeJf36POk6yJV_adQssw5c",
"consents_token":"eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY
3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiaWF0IjoxNTE2MjM5MDIyfQ.SflKxwRJSMeK
KF2QT4fwpMeJf36POk6yJV_adQssw5c"

}

2.7 Response

If the find request sent from the PFS succeeds, the data provider (UMA resource server) will respond with a *202 Accepted HTTP* status implying acknowledgment of the find request. Following this the PFS will not be involved in any request back from the data provider.

HTTP/1.1 202 Accepted

If a match is found, then the data provider will issue a PeI in the agreed format alongside its description and register this directly to the consent and authorisation service, via the UMA protection API. If no match is found, then there is no further action required by the data provider other than the option to hash failed find requests, in order to avoid repeating them. This optional requirement is discussed in more detail in the data provider technical guide.

2.8 Error handling

If the find request sent from the PFS fails, then the data provider is expected to respond with the appropriate HTTP status code and error message. The body of the response for an error must contain an error code that correlates to a message, so that the test harnesses and logs can be automated

4xx
The 4xx class of status code is intended for cases in which the client seems to have erred. Except when responding to a POST request, the server SHOULD include an entity containing an explanation of the error situation, and whether it is a temporary or permanent condition.

5xx
The 5xx (server error) class of status code indicates that an exception occurred during the elaboration of a request. An indication about the nature of the error SHOULD be included together with an indication if the error is temporary or permanent.

Status	Message	Note
500	internal server error	
503	service unavailable	This status is used temporarily when the service is down for maintenance or is overloaded by requests.

3 Obtain PAT API

3.1 Summary of the obtain PAT API

During find, the PFS includes a valid user account token issued by the authorisation server (part of the C&A) in the find request sent out to the data provider find interface endpoints. The user account token is an OAuth2 authorisation grant, expressed as a JWT (JSON web token) that can be exchanged for the PAT, which is an OAuth2 access token as per the OAuth2 standard, by the presentation of this token to the authorisation server's token endpoint. Data providers will be a client of the authorisation server.

Hosting
The API will be hosted on the authorisation server (AS) as the request is made is to the authorisation server's token endpoint.

3.2 Format

The obtain PAT API will be a REST API with character encoding of UTF-8 in the HTTP request entity-body.

3.3 Authorisation

The data provider will be a client of the authorisation server and will need to register its software with it before a request can be made. This will ensure the communication channel is encrypted and will establish dynamic trust between both parties.

3.4 HTTP method

The data provider MUST use the HTTP "POST" method when making access token requests to the authorisation server's token endpoint, by sending the following parameters using the "application/x-www-form-urlencoded" format with a character encoding of UTF-8 in the HTTP request entity-body:

grant_type
REQUIRED. Value MUST be set to " urn:ietf:params:oauth:grant-type:jwt-bearer".

assertion
REQUIRED. MUST contain a single JWT

scope
REQUIRED. Specifies the scope of the access request. MUST be uma_protection

For example, the data provider makes the following HTTP POST request using mTLS:

```
POST /token HTTP/1.1
    Host: as.pdp.com
    Content-Type: application/x-www-form-urlencoded;charset=UTF-8

    grant_type=urn%3Aietf%3Aparams%3Aoauth%3Agrant-type%3jwt-bearer&
    assertion+ PHNhbWxwOl...[omitted for brevity]...ZT4&scope= uma_protection
```

3.5 Response

If the request for an access token is valid, the authorisation server generates an access token (PAT) as a structured JWT bound (in accordance with Section 3 RFC8705) to the resource server (data provider certificate).

For example, a successful token response may look like the following with all the properties of the PAT token encoded in the payload:

```
HTTP/1.1 200 OK
Content-Type: application/json;charset=UTF-8

{
“access_token”:“eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3
ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiaWF0IjoxNTE2MjM5MDIyfQ.SflKxwRJSMeKK
F2QT4fwpMeJf36POk6yJV_adQssw5c”,
“token_type”:“pension_dashboard_pat”
}
```

3.6 Error handling

If the access token request is invalid, such as the redirect URL didn’t match the one used during authorisation, then the server will return an error response. The error handling will follow standard OAuth 2 failure codes as per RFC6749.

Error responses are returned with an HTTP 400 status code (unless specified otherwise), with error and error_description parameters. The error parameter will always be one of the values listed below:

- *invalid_request – the request is missing a parameter so the server can't proceed with the request. This may also be returned if the request includes an unsupported parameter or repeats a parameter*
- *invalid_grant – the authorisation code (or user's password for the password grant type) is invalid or expired. This is also the error you would return if the redirect URL given in the authorisation grant does not match the URL provided in this access token request*

- *invalid_scope – for access token requests that include a scope (password or client_credentials grants), this error indicates an invalid scope value in the request*
- *unauthorised_client – this client is not authorised to use the requested grant type. For example, if you restrict which applications can use the Implicit grant, you would return this error for the other apps*
- *unsupported_grant_type – if a grant type is requested that the authorisation server doesn't recognise, use this code. Note that unknown grant types also use this specific error code rather than using the invalid_request above.*

The entire error response is returned as a JSON string, similar to the successful response. Below is an example of an error response.

```
HTTP/1.1 400 Bad Request
Content-Type: application/json;charset=UTF-8
Cache-Control: no-store
Pragma: no-cache

 {

"error": "invalid_request",

}
```

4 Register PeI API

4.1 Summary of the register PeI API

When the data provider (UMA resource server) has determined a match within their internal records and obtained the PAT token to enable access to the UMA protection API hosted in the authorisation server, then the data provider will be able to access the API to register PeIs (resources) and place them under protection of an authorisation control on behalf of the pension owner (resource owner) and manage them over time. Protection of a resource at the authorisation server begins on successful registration and ends on successful deregistration.

The authorisation server MUST support the following five registration options and MUST require a valid PAT for access to them; any other operations are undefined by this specification. Here, rreguri stands for the resource registration endpoint and _id stands for the authorisation server-assigned identifier for the web resource, returned by the authorisation server when the create resource operation was performed, corresponding to the resource at the time it was created, included within the URL returned in the location header. Each operation is defined in its own section below.

Create resource description: POST rreguri/
Read resource description: GET rreguri/_id
Update resource description: PUT rreguri/_id
Delete resource description: DELETE rreguri/_id
List resource descriptions: GET rreguri/

The resource server must persist the following, for each resource owner following a create resource operation:

- resource _id – index of the registered resource (PeI)
- resource owner's PAT – access token to API
- authorisation server 'AS URI' which issued the PAT (at which the resource _id is registered) - address of the authorisation server token endpoint

The resource server should also persist these items in a manner which it can locate them using the inbound URL of the view request.

4.2 Hosting

The API will be hosted on the authorisation server, which is part of the consent and authorisation (C&A) service. The resource server will use this API at the authorisation server's resource registration endpoint to create, read, update and delete resource descriptions, along with retrieving lists of such descriptions. The descriptions consist of JSON documents that are maintained as web resources at the authorisation server. The authorisation server should declare this endpoint in the discovery document, so that the resource server knows the endpoint.

4.3 Format

The API will be a REST API using JSON encoded as UTF-8.

4.4 Authorisation

The data provider will need to use the relevant PAT specific to the pension owner to authorise its use of the API when making a request to the authorisation server to create, read, update or delete a resource (PeI).

4.5 HTTP method

This will depend on the type of request ie create, read, update, delete, list. These are covered in the relevant sections below.

4.6 Resource description

A resource description is a JSON document that describes the characteristics of a resource sufficiently for an authorisation server to protect it. A resource description will have the following parameters:

Resource scopes – required
These will be ["value", "owner", "delegate"]. All three scopes must be used for all registrations, so that the resource owner can subsequently delegate access if required without further resource server activity.

Name – required
This is the URI or URN of the resource, ie of the pension asset at the resource server. It will be used as the unique name against which the authorisation server will apply authorisation protection and is the representation of the pension asset as is available to a pensions dashboard client.

Type – required
This is a URI of the type of all 'name' parameters used in the pensions dashboards ecosystem. It is required for future extension, eg to support resources which have wider scope options than defined here, or for specialised RS-AS relationships in the future.

Description – required
Data standards for the pensions dashboards ecosystem will define 'type' and 'name' mandated here. It may be that human readable derivations of 'name' will be aided by this 'description'. Format of this field needs to be agreed with the pensions industry.

4.6.1 Create resource description

The resource server must use the HTTP "POST" method when registering the resource with the authorisation server. The resource server must register each pension asset as a separate resource (to enable delegation and access at the most granular level). The request must contain the required parameters.

Example of a resource registration request message with a PAT in the header:

```
POST /rreg/ HTTP/1.1
Content-Type: application/json;charset=UTF-8
Authorisation: Bearer
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI
6IkpvaG4gRG9lIiwiaWF0IjoxNTE2MjM5MDIyfQ.SflKxwRJSMeKKF2QT4fwpMeJf36P
Ok6yJV_adQssw5c

...
{
  "resource_scopes":["value", “owner”, “delegate”],
  "name": "pei:aed123aq:WGF45920EJH348ASEWQ0284",
  "type": “http://pdp.gov/uma/PEI”,
  “description” : “Aviva Pension 1234”
}
```

If the request is successful, the resource is thereby registered and the authorisation server MUST respond with an HTTP 201 status message that includes a location header and a resource_id parameter. The resource _id parameter is issued by the authorisation server for each registered resource, ie as a result of each UMA registration of a PeI, initiated by the resource server, authorised by the PAT. The resource_id is the common index between the authorisation server and resource server associated with each PeI.

Example resource registration response message:

```
HTTP/1.1 201 Created
Content-Type: application/json;charset=UTF-8
Location: /rreg/KX3A-39WE
...
{
  " resource_id":"KX3A-39WE"

}
```

4.6.2 Read resource description

The resource server must use the HTTP GET method to read a previously registered resource description. If the request is successful, the authorisation server MUST respond with an HTTP 200

status message that includes a body containing the referenced resource description, along with an _id parameter.

Example of a read request, with a PAT in the header:

```
GET /rreg/KX3A-39WE HTTP/1.1
Authorisation: Bearer
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI
6IkpvaG4gRG9lIiwiaWF0IjoxNTE2MjM5MDIyfQ.SflKxwRJSMeKKF2QT4fwpMeJf36P
Ok6yJV_adQssw5c
...
```

Example of a successful response, containing all the parameters that were registered as part of the description:

```
HTTP/1.1 200 OK
Content-Type: application/json;charset=UTF-8
...
{
  " resource_id":"KX3A-39WE",
  "resource_scopes":["value", “owner”, “delegate”],
  "name": "pei:aed123aq:WGF45920EJH348ASEWQ0284",
  "type": “http://pdp.gov/uma/PEI”,
  “description” : “Aviva Pension”
}
```

4.6.3 Update resource description

The resource server must use the HTTP PUT method to update a previously registered resource description, by means of a complete replacement of the previous resource description. If the request is successful, the authorisation server MUST respond with an HTTP 200 status message that includes a resource_id parameter.

Example of an update request adding a description parameter to a resource description that previously had none, with a PAT in the header:

```
PUT /rreg/9UQU-DUWW HTTP/1.1
Content-Type: application/json
Authorisation: Bearer
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI
6IkpvaG4gRG9lIiwiaWF0IjoxNTE2MjM5MDIyfQ.SflKxwRJSMeKKF2QT4fwpMeJf36P
Ok6yJV_adQssw5c
...
{
  "resource_scopes":["value”, “owner”, “delegate”],
  "name": " pei: aed123aq:WGF45920EJH348ASEWQ0284",
```

```
    "type": “http://pdp.gov/uma/PEI”,
     “description” : “Aviva Pension”
  }
```

Form of a successful response, not containing the optional user_access_policy_uri parameter:

```
HTTP/1.1 200 OK
...
{
  "resource_id":"9UQU-DUWW"
}
```

4.6.4 Delete resource description

The resource server must use the HTTP Delete method to delete a previously registered resource description. If the request is successful, the resource is thereby deregistered and the authorisation server MUST respond with an HTTP 200 or 204 status message.

Form of a delete request, with a PAT in the header:

```
DELETE /rreg/9UQU-DUWW
Authorisation: Bearer
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI
6IkpvaG4gRG9lIiwiaWF0IjoxNTE2MjM5MDIyfQ.SflKxwRJSMeKKF2QT4fwpMeJf36P
Ok6yJV_adQssw5c
...
```

Form of a successful response:

```
HTTP/1.1 204 No content
...
```

4.6.5 List resource description

The resource server must use the HTTP GET method in order to get a list of all previously registered resource identifiers for this resource owner. The authorisation server MUST return the list in the form of a JSON array of resource_id string values.

The resource server can use this method as a first step in checking whether its understanding of protected resources is in full synchronisation with the authorisation server's understanding.

Form of a list request, with a PAT in the header:

```
GET /rreg/ HTTP/1.1
Authorisation: Bearer
```

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI
6IkpvaG4gRG9lIiwiaWF0IjoxNTE2MjM5MDIyfQ.SflKxwRJSMeKKF2QT4fwpMeJf36P


Ok6yJV_adQssw5c
...
```

Form of a successful response:

```
HTTP/1.1 200 OK
...
[
  "KX3A-39WE",
  "9UQU-DUWW"
]
```

4.7 Error handling

If the request fails because the resource server does not have a valid access token (ie the PAT is missing or has expired) then the authorisation server responds with an HTTP 401 status code as the request cannot be authenticated.

If a request is successfully authenticated, but is invalid for another reason, the authorisation server produces an error response by supplying a JSON-encoded object with the following members in the body of the HTTP response:

error – REQUIRED except as noted. A single error code. Values for this parameter are defined throughout this specification.

error_description – OPTIONAL - human-readable text providing additional information.

error_uri – OPTIONAL - a URI identifying a human-readable web page with information about the error.

```
HTTP/1.1 400 Bad Request
Content-Type: application/json
Cache-Control: no-store
...

{
  "error": "invalid_resource_id",
  "error_description": "Permission request failed with bad resource ID.",

  "error_uri": "errors to be defined later in design"
}
```

If the request to the resource registration endpoint is incorrect, then the authorisation server instead responds as follows:

If the referenced resource cannot be found, the authorisation server MUST respond with an HTTP 404 (not found) status code and MAY respond with a not_found error code.

If the resource server request used an unsupported HTTP method, the authorisation server MUST respond with the HTTP 405 (Method not allowed) status code and MAY respond with an unsupported_method_type error code.

If the request is missing a required parameter, includes an invalid parameter value, includes a parameter more than once, or is otherwise malformed, the authorisation server MUST respond with the HTTP 400 (Bad request) status code and MAY respond with an invalid_request error code.

5 View API

5.1 Summary of view API

Enables a dashboard (client) to retrieve pension details on behalf of a requesting party (ie pension owner or delegate) by dereferencing the PeI which resolves to a URL and making a HTTP GET request to access the pension details. It is this URL which is an UMA protected resource and if the request is authorised via the UMA protocol then the data provider will respond back to the dashboard with the pension details encoded within the data payload as a JWT.

5.2 Hosting

Each data provider connected to the ecosystem will be required to host their view API within their domain.

5.3 Format

The view API will be a REST API using JSON encoded as UTF-8.

5.4 Authorisation

The dashboard client will authenticate itself with the authorisation server at run time as defined in rfc8705 section 2 . In addition to this, all end point connections are secured using mutual TLS.

5.5 HTTP method

The dashboard will make a HTTP GET request to the data provider's view endpoint by dereferencing the PeI, which resolves into a URL. The request will carry sufficient information to identify the resource owner at the data provider and the type of access being attempted:

- *an identifier for the 'pension resource' owned by the resource owner at the resource server, which is being accessed*
- *the type of requesting party (owner or delegate)*

The assumed design of the 'unique dereferenceable identifier' for each individual pension resource is of the following form, and the access can carry a query parameter asserting the nature of the requesting party user:

- *'pei':<holdername>:<assetidentifer>*
- *?user=owner (the default if absent) or ?user=delegate (for advisers or guidance staff)*

The asset universally unique identifier (UUID) can be used as a key within the data provider to locate both the internal asset and the pension owner (ie resource owner's identifier and PAT) with which it is associated and persisted by the data provider after the create resource operation. The data provider will use the PAT associated with the pension owner to coordinate with the authorisation server to introspect the requesting party token (RPT – see section 6) or to obtain a permissions ticket (section 7) if necessary.

Example of a view request carrying an RPT:

```
GET/pei/aed123aq/WS23JQ48KH789KS349?user=owner HTTP/1.1
Host: www.dashboard.aviva.com
Authorisation: Bearer
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI
6IkpvaG4gRG9lIiwiaWF0IjoxNTE2MjM5MDIyfQ.cThIIoDvwdueQB468K5xDc5633se
EFoqwxjF_xSJyQQ.
```

In the above example the asset UUID is WS23JQ48KH789KS349.

5.6 Response

If the view request has been determined to be authorised (as a result of introspecting the RPT or the cached result of previous introspection), then the data provider will respond back to the dashboard with the pension details encoded within the data payload as defined in the data standards for view as a JWT.

Example of a successful response:

```
HTTP/1.1 200 OK
Content-Type: application/json;charset=UTF-8
```

...
{

“pension_details”:”eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJwZW5zaW9uQXJyY
W5nZW1lbnREZXRhaWxzIjp7InBlbnNpb25SZWZlcmVuY2UiOiIwMDAzNzQ2MiIsInB
lbnNpb25OYW1lIjoiU09MQVIgRU5FUkdZIFNZU1RFTVMgUEVOU0lPTiBGVU5EIiwicG
Vuc2lvblR5cGUiOiJEQiIsInBlbnNpb25PcmlnaW4iOiJXIiwicGVuc2lvblN0YXR1cyI6IkE
iLCJwZW5zaW9uU3RhcnREYXRlIjoiMjAwNC0xMC0yMyIsInBlbnNpb25SZXRpcmVtZ
W50RGF0ZSI6IjIwNDUtMDctMDYifX0.G7oYVo1d18N-Y1TVBN-
db9oruAjQNHureeM62hsPvOc”

}

5.7 Error handling

If the view request is invalid the data provider will respond back to the dashboard with an HTTP 401 code and an appropriate error message.

If the view request is missing an RPT or has an invalid RPT, then the data provider in its response will provide a WWW-Authenticate header with the authentication scheme UMA, with the issuer URI from the authorisation server's discovery document in an as_uri parameter indicating the URL of the authorisation server where the dashboard should reach for further interactions to get an access token and the permission ticket in a ticket parameter. This will enable the dashboard to initiate the authorisation protocol and obtain a valid RPT with the authorisation server.

For example:

HTTP/1.1 401 Unauthorized
WWW-Authenticate: UMA realm="PensionDashboard",
as_uri="https://as.pdp.com",

ticket=”eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiw
ibmFtZSI6IkpvaG4gRG9lIiwiaWF0IjoxNTE2MjM5MDIyfQ.cThIIoDvwdueQB468K5
xDc5633seEFoqwxjF_xSJyQQ”
...

If the data provider is unable to provide a permission ticket from the authorisation server or introspect the RPT with the authorisation server, as it is unable to access the protection API because the PAT has expired, then it includes an error message telling the dashboard the PAT has expired and needs to be refreshed.

For example:

HTTP/1.1 401 Unauthorized
as_uri="https://as.pdp.com",

```
    error: "PAT expired"
```

Dashboards which have complied with the redirection and UMA protocols must not repeatedly retry and simply inform their user of remedial action, as such a state is possible if the user has withdrawn consent for this dashboard, or their state at the C&A is indeterminate (they haven't proved their identity) or no PeIs are shareable.

6 Introspect API

6.1 Summary of introspect API

When a resource server receives a view request from a dashboard, which is accompanied by the access token (RPT), the resource server will need to determine whether the access token is active and, if so, its associated permissions before any pension details can be retrieved and sent back to the dashboard. It does this by introspecting the RPT at the authorisation server by using the introspect API. The response of the introspection can be cached for an appropriate amount of time, yet to be defined by PDP, so that if the same view request is made during the validity of the cached response, then the resource server does not need to make a repeat request to the authorisation server to check whether the RPT is active. It can determine this by using a cached copy of the token introspection response. This will avoid excessive load on the authorisation server.

6.2 Hosting

The API will be hosted on the authorisation server, which is part of the C&A service. The authorisation server will declare this endpoint in the discovery document, so that the resource server knows the endpoint.

6.3 Format

The API will be a REST API using JSON encoded as UTF-8.

6.4 Authorisation

The data provider will need to use the relevant PAT, specific to the pension owner, to authorise its use of the API, when making an introspection request to the authorisation server.

6.5 HTTP method

The resource server will call the Introspection API using HTTP POST method.

Example of the resource server's request to the authorisation server for introspection of an RPT, with a PAT in the header:

```
POST /introspect HTTP/1.1
Host: as.pdp.com
Content-Type: application/x-www-form-urlencoded;charset=UTF-8
Authorization: Bearer
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI
6IkpvaG4gRG9lIiwiaWF0IjoxNTE2MjM5MDIyLCJhdWQiOiIxMjIzNDg5MCIsImV4cCI
6MTIzNDU2Nzh9.XliAWL97BMJx4V3WlIZESvEWhw7DGGUTJAOpIGYv_H0
...
token=eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3OdkwIiwib
mFtZSI6IkpvaG4gRG9lIiwiaWF0IjoxNTE2MjM5MDIyfQ.cThIIoDvwdueQB468K5xDc
5633seEFoqwxjF_xSJyQQ
```

6.6 Response

The authorisation server responds with a JSON object in "application/json" format, with the following parameters in the payload encoded in a JWT:

resource_id
REQUIRED. A string that uniquely identifies the protected resource, access to which has been granted to this client on behalf of this requesting party. The identifier MUST correspond to a resource that was previously registered as protected.

Resource_scopes
REQUIRED. An array referencing zero or more strings representing scopes to which access was granted for this resource. Each string MUST correspond to a scope that was registered by this resource server for the referenced resource.

iss
REQUIRED. Registered claim name. Defined [JWT]. Profiled: unique identifier within dashboard ecosystem of the AS issuing the JWT.

sub
REQUIRED. Registered claim name. Defined [JWT]. Profiled: unique identifier within scope of iss of the requesting party, which was assured by the AS when the RPT was issued. *(This is the assured user as known at the AS, based on the PCT or on the contents of a PMT during step-up authentication.)*

aud
REQUIRED. Registered claim name. Defined [JWT]. Profiled: unique identifier within the scope of the

dashboard ecosystem of the resource server. *(Derived from the PAT used in the initial permission ticket request, via the PMTs.)*

iat
REQUIRED. Registered claim name. Defined [JWT]. Profiled: time of issue.

exp
REQUIRED. Registered claim name. Defined [JWT]. Profiled: time of expiry.

jti
REQUIRED. Registered claim name. Defined [JWT]. Profiled: using jti as the unique token identifier.

Public claim name. *none*

rqp
REQUIRED. Private claim name. Structured representation (JSON object) of the *pension_dashboard_rqp* which was presented with the permission ticket in the successful authorisation call to the AS.

permission
REQUIRED. Private claim name. UMA permission as granted by the AS following the permission request represented in the PMT defined above.

owner
OPTIONAL. Private claim name. The identifier of the resource owner at the resource server. *This is from the PMT, initially from PAT. May be of use to the resource server to minimise lookup time of resource _id to derive the owner of the resource.*

The token must be signed by the issuer. It SHOULD be encrypted for the RS.

Example of a response containing the introspection object within an encoded JWT:

```
HTTP/1.1 200 OK
Content-Type: application/json;charset=UTF-8
...
{
“token”:“eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiw
ibmFtZSI6IkpvaG4gRG9lIiwiaWF0IjoxNTE2MjM5MDIyLCJhdWQiOiIxMjIzNDg5MCIs
ImV4cCI6MTIzNDU2Nzh9.XliAWL97BMJx4V3WlIZESvEWhw7DGGUTJAOpIGYv_H0
“
}
```

6.7 Error handling

If the request to the introspection endpoint is incorrect, then the authorisation server instead responds as follows:

If the referenced resource cannot be found, the authorisation server MUST respond with an HTTP 404 (not found) status code and MAY respond with a not_found error code.

If the resource server request used an unsupported HTTP method, the authorisation server MUST respond with the HTTP 405 (method not allowed) status code and MAY respond with an unsupported_method_type error code.

If the request is missing a required parameter, includes an invalid parameter value, includes a parameter more than once, or is otherwise malformed, the authorisation server MUST respond with the HTTP 400 (bad request) status code and MAY respond with an invalid_request error code.

7 Permission API

7.1 Summary of permission API

If the view request made by the dashboard is without an RPT or is accompanied by an invalid RPT, then the resource server will coordinate with the authorisation server to request one or more permissions (resource identifiers and corresponding scopes) on the dashboard's behalf and receive a permissions ticket on return. The resource server must request scopes "value" and either "delegate" (if the inbound call explicitly requested this) or "owner" (by default or as explicitly requested), but not both. The permissions ticket is used by the dashboard to initiate the UMA Grant protocol with the authorisation server to obtain a new RPT in order to authorise its view request.

7.2 Hosting

The API will be hosted on the authorisation server, which is part of the C&A service. The authorisation server will declare this endpoint in the discovery document, so that the resource server knows the endpoint.

7.3 Format

The API will be a REST API using JSON encoded as UTF-8.

7.4 Authorisation

The data provider will need to use the relevant PAT, specific to the pension owner, to authorise its use of the API when making permissions requests to the authorisation server.

7.5 HTTP method

The resource server will call the permission API using the HTTP POST method. The body of the HTTP request message contains a JSON object for requesting a permission for a single resource identifier.

The object format is derived from the resource description format specified in Section 4.6; it has the following parameters:

resource_id
REQUIRED. The identifier for a resource to which the resource server is requesting a permission on behalf of the client. The identifier MUST correspond to a resource that was previously registered.

resource_scopes
REQUIRED. An array referencing zero or more identifiers of scopes to which the resource server is requesting access for this resource on behalf of the client. Each scope identifier MUST correspond to a scope that was previously registered by this resource server for the referenced resource.

Example of an HTTP request for a single permission at the authorisation server's permission endpoint, with a PAT in the header:

```
POST /perm HTTP/1.1
Content-Type: application/json;charset=UTF-8
Host: as.pdp.com
Authorization: Bearer
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI
6IkpvaG4gRG9lIiwiaWF0IjoxNTE2MjM5MDIyLCJhdWQiOiIxMjIzNDg5MCIsImV4cCI
6MTIzNDU2Nzh9.XliAWL97BMJx4V3WlIZESvEWhw7DGGUTJAOpIGYv_H0
...

{
  "resource_id":" KX3A-39WE",
   "resource_scopes":["value", “owner”],
}
```

7.6 Response

If the authorisation server is successful in creating a permission ticket in response to the resource server's request, it responds with an HTTP 201 (created) status code and includes the ticket

parameter in the JSON-formatted body as a signed JWT. Regardless of whether the request contained one or multiple permissions, only a single permission ticket is returned.

For example:

```
HTTP/1.1 201 Created
Content-Type: application/json;charset=UTF-8
...

{
“ticket”:”eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiw
ibmFtZSI6IkpvaG4gRG9lIiwiaWF0IjoxNTE2MjM5MDIyLCJhdWQiOiIxMjIzNDg5MCIs


ImV4cCI6MTIzNDU2Nzh9.XliAWL97BMJx4V3WlIZESvEWhw7DGGUTJAOpIGYv_H”
}
```

The PMT token is a JWT[1] as defined in [JWT] and contains the following claims:

iss
REQUIRED. Registered claim name. Defined [JWT]. Profiled: unique identifier within the dashboard ecosystem of the authorisation server) issuing the JWT.

sub
REQUIRED. Registered claim name. Defined [JWT]. Profiled: unique identifier within the scope of iss of the resource owner identifier at the authorisation server *(derived from the PAT used in the initial permission ticket request).*

aud
REQUIRED. Registered claim name. Defined [JWT]. Profiled: unique identifier within the scope of the dashboard ecosystem of the authorisation server.

iat
REQUIRED. Registered claim name. Defined [JWT]. Profiled: time of issue.

exp
REQUIRED. Registered claim name. Defined [JWT]. Profiled: time of expiry.

jti
REQUIRED. Registered claim name. Defined [JWT]. Profiled: using jti as the unique token identifier.

Public claim name. *none*

[1] PDP's profile of UMA suggests the use of structured tokens however this is not mandatory and is up to the choice of the UMA implementors

rs
REQUIRED. Private claim name. The identifier of the resource server. *(Derived from the PAT used in the initial permission ticket request.)*

owner
OPTIONAL. Private claim name. The identifier of the resource owner at the resource server. *(Derived from the PAT used in the initial permission ticket request.) May be of use to the resource server to minimise lookup time of resource _id to derive the owner of the resource.*

permissions
REQUIRED. Private claim name. UMA permission as defined in [UMAGrant] and [UMAFedAuthz] using scopes defined in this profile section 7.1.

rqp
OPTIONAL. Private claim name. Structured representation (JSON object) of the *pension_dashboard_rqp* which was presented with the permission ticket (if any) in a previous call to the authorisation server (AS). Claim must be present when the permission ticket is presented for the second or subsequent time by a dashboard client. The authorisation server must populate this claim with the contents of the *pension_dashboard_rqp* which was presented in the call for which it is reissuing a permission ticket, having checked that the content is in accord with the same requesting party presenter.

assuredID
OPTIONAL. Private claim name. Structured representation (JSON object) of the identity of the requesting party (as uniquely represented at the authorisation server (AS)) and the asserting identity provider reference. Claim is present when the AS reissues it after assured identification and if necessary, confirmation of the assured professional status of a 'delegate' requesting party.

The token must be signed by the issuer. It MUST be encrypted for the authorisation server.

The token may be persisted by the authorisation server to enable correlation across presentations of such tokens.

As per [UMAGrant] 5.5 permission tickets are single use: the AS must issue a new token with a new jti for every iteration of the permission process. The AS must ensure that authorisation process and any dependent tokens are revoked if a permission ticket is replayed.

The token will always be presented to the token or claims interaction endpoints at the AS by the dashboard client so it does not need to be bound further than application level checking by the AS which must ensure that the pension_dashboard_rqp details match across related calls.

7.7 Error handling

If the resource server's permission registration request is authenticated properly but fails due to other reasons, the authorisation server responds with an HTTP 400 (bad request) status code and includes one of the following error codes:

invalid_resource_id – At least one of the provided resource identifiers was not found at the authorisation server.

invalid_scope – At least one of the scopes included in the request was not registered previously by this resource server for the referenced resource.

8 PAT refresh API

8.1 Summary of PAT refresh API

When the PAT needs to be refreshed, the authorisation server will need to send across the required parameters to the data provider in order for them to be able to obtain a new PAT. The authorisation server will call this API, which will be exposed by data providers. They will receive the required parameters and trigger them to coordinate with the authorisation server, to exchange the temporary credential user account token, which is an OAuth authorisation grant for the PAT.

8.2 Hosting

Each data provider connected to the ecosystem will be required to host their PAT refresh API within their domain.

8.3 Format

The refresh API will be an OAuth REST API using JSON encoded as UTF-8.

8.4 Authorisation

This is a closed ecosystem with all end point connections secured using mutual TLS, with only the authorisation server invoking the PAT refresh endpoints. There is an assumption made by the Pensions Dashboards Programme that there are no additional API security requirements for the refresh API – this needs to be validated with the industry.

8.5 HTTP method

The authorisation server will be restricted to only making an HTTP POST request to each data provider PAT refresh endpoint containing the required parameters: the user account token, which is a OAuth2 authorisation grant expressed as a JWT that can be exchanged for the PAT, the resource owner's consent expressed as a JWT , the resource_id and PeI, so that the data provider can locate the resource owner and PAT which requires refreshing.

For example, the consent and authorisation service makes the following HTTP POST request using mTLS:

```
POST /refresh HTTP/1.1
Host: www.dashboard.aviva.com
Content-Type: application/json;charset=UTF-8
{

“user_account_token”:”eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM
0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiaWF0IjoxNTE2MjM5MDIyfQ.SflKxwRJS
SflKxwRJSMeKKF2QT4fwpMeJf36POk6yJV_adQssw5c”,
"consents_token":"eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY
3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiaWF0IjoxNTE2MjM5MDIyfQ.SflKxwRJSMeK
SflKxwRJSMeKKF2QT4fwpMeJf36POk6yJV_adQssw5c",
"resource_id": “KX3A-39WE”,
“pei”:“pei:aed123aq:WGF45920EJH348ASEWQ0284"

}
```

8.6 Response

If the request sent by the consent and authorisation service is successful, then the data provider will respond with an HTPP *202 Accepted* status implying acknowledgment of the PAT refresh request.

HTTP/1.1 202 Accepted

Following that, the resource server requests a new PAT by quoting the user account token in its request to the authorisation server token endpoint – see Section 3 Obtain PAT.

8.7 Error handling

If the PAT refresh request sent by the authorisation server fails, then the data provider is expected to respond with the appropriate HTTP status code and error message. The body of the response for an error must contain a message in plain language that describes, accordingly with the error catching process, what went wrong and possibly how to amend it, so as to issue a new, valid request.

4xx
The 4xx class of status code is intended for cases in which the client seems to have erred. Except when responding to a POST request, the server SHOULD include an entity containing an explanation of the error situation, and whether it is a temporary or permanent condition.

Status	Message	Note
400	bad request	Potential bad requests examples: • *authorisation server is not sending a HTTP POST* • *parsing error by data provider* • *schema not configured correctly*

5xx
The 5xx (server error) class of status code indicates that an exception occurred during the elaboration of a request. An indication about the nature of the error SHOULD be included together with an indication if the error is temporary or permanent.

Status	Message	Note
500	internal server error	

9 Authorise API

9.1 Summary of authorise API

The dashboard initiates the authorisation 'dance' using this OAuth specific API to authorisation server (AS) token endpoint, to obtain a new access token (RPT). The dashboard will need to provide a set of claims needed for the AS to assess the current authorisation context when interacting with the AS token endpoint. Once this process is successful, the AS in its response will issue a new access token (RPT) for the dashboard, to authorise its view requests in the future.

9.2 Hosting

The API will be hosted on the authorisation server, which is part of the consent and authorisation service. The dashboard will have obtained the as_uri from the resource server following a failed view request.

9.3 Format

The API will be a REST API using JSON encoded as UTF-8.

9.4 Authorisation

The dashboard client will authenticate itself with the authorisation server at run time, as defined in rfc8705 section 2 . In addition to this, all end point connections are secured using mutual TLS.

9.5 HTTP method

The dashboard MUST use the HTTP "POST" method when making access token requests to the authorisation server's token endpoint. It sends the following parameters using the "application/x-www-form-urlencoded" format, with a character encoding of UTF-8 in the HTTP request entity-body:

grant_type
REQUIRED. MUST be the value urn:ietf:params:oauth:grant-type:uma-ticket.

ticket
REQUIRED. The most recent permission ticket received by the client (dashboard) as part of this authorisation process. Permission ticket is a structured JWT.

claim_token
REQUIRED. The client must provide a claim of type pension_dashboard_rqp. The contents of this token must represent the current requesting party user at the dashboard client.

claim_token_format
REQUIRED. The claim (above) is of a specific type 'pension_dashboad_rqp'. UMA requires this parameter to match the claim(s).

scope
REQUIRED. The dashboard client request MUST contain the requested pension dashboard scopes: value and owner/delegate

pct
OPTIONAL. The client MUST provide its existing PCT for the requesting party (ie for the combination of the client and the its user), for the resource the client is seeking to access for that requesting party, if it has one, even if it knows that the PCT has expired.

rpt
OPTIONAL. The client SHOULD provide its existing RPT for the resource it requested in the previous call to the same resource server for the same requesting party, if it has one, even if it knows that this RPT is expired. *(Although the profile will not upgrade RPTs, this is included to enable flexibility for possible future extensions.)*

For example, the dashboard client makes the following HTTP POST request using mTLS:

```
POST /token HTTP/1.1
Host: as.pdp.com
Content-Type: application/x-www-form-urlencoded;charset=UTF-8
...
```

grant_type=urn%3Aietf%3Aparams%3Aoauth%3Agrant-type%3Auma-ticket
&ticket=eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZ
SI6IkpvaG4gRG9lIiwiaWF0IjoxNTE2MjM5MDIyfQ.SflKxwRJSMeKKF2QT4fwpMeJf36POk6y
JV_adQssw5c
&claim_token= aGVsbG8gd29ybGQ
&claim_token_format= *this type name needs agreed URI format, based on the profile's domain*
&scope=value owner
&pct=eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI
6IkpvaG4gRG9lIiwiaWF0IjoxNTE2MjM5MDIyLCJleHAiOjM1Nzg5Nzg5NywianRpIjo1NDIzNj
IzOTgwfQ.hp1udPrSdTRkLf3QzRoHFff_T6BzRqlSADIZoZ-c5sI

9.6 Response

If the permission request is successful the authorisation server will issue the RPT and optionally a PCT if required.

HTTP/1.1 200 OK
Content-Type: application/json;charset=UTF-8
...

{

"access_token":"eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIi
wibmFtZSI6IkpvaG4gRG9lIiwiaWF0IjoxNTE2MjM5MDIyfQ.SflKxwRJSMeKKF2QT4fwpMeJf
SflKxwRJSMeKKF2QT4fwpMeJf36POk6yJV_adQssw5c ",
“token_type”: “pension_dashboard_rpt”,
“upgraded”: false,
"pct":"eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI
6IkpvaG4gRG9lIiwiaWF0IjoxNTE2MjM5MDIyLCJleHAiOjM1Nzg5Nzg5NywianRpIjo1NDIzNj
IzOTgwfQ.hp1udPrSdTRkLf3QzRoHFff_T6BzRqlSADIZoZ-c5sI"

}

9.7 Error handling

If the permission request is unsuccessful the authorisation server will respond with the appropriate error.

Example of a need_info response, with a hint to redirect the requesting party to a claims interaction endpoint:

HTTP/1.1 403 Forbidden
Content-Type: application/json;charset=UTF-8
Cache-Control: no-store

```
...

{

        "error":"need_info",
        “ticket":"eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibm
        FtZSI6IkpvaG4gRG9lIiwiaWF0IjoxNTE2MjM5MDIyLCJleHAiOjM1Nzg5Nzg5NywianRpIjo
        1NDIzNjIzOTgwfQ.hp1udPrSdTRkLf3QzRoHFff_T6BzRqlSADIZoZ-c5sI ",
        "redirect_user":https://as.pdp.com/rqp_claims?id=2346576421

}
```

Example when the client was not authorised to have the permissions:

```
HTTP/1.1 403 Forbidden
Content-Type: application/json;charset=UTF-8
Cache-Control: no-store
...

{
  “error”:”request_denied”
}
```

10 Obtain PeIs API

10.1 Summary of obtain PeIs API

In order for the dashboard, whether used by the owner or their delegate, to obtain the owner’s PeIs following the completion of find, the dashboard will pull the PeIs from the owner user’s consent and authorisation account, via an API hosted at the authorisation server.

10.2 Hosting

The API (an UMA Resource Server) will be hosted on the authorisation server, which is part of the consent and authorisation service.

10.3 Format

The API will be a REST API using JSON encoded as UTF-8.

10.4 Authorisation

The dashboard client will authenticate itself with the authorisation server at run time as defined in rfc8705 section 2 . In addition to this, all end point connections are secured using mutual TLS.

10.5 HTTP method

The dashboard will make a HTTP GET request to the obtain PeI endpoint for that specific user's consent and authorisation account – ie the dashboard attempts to GET the URL https:://CA.ObtainPeIs/<userGUID> with a parameter of the user is 'owner' or 'delegate'. The 'userGUID' is the unique identifier for the user's consent and authorisation account. As it is a protected resource the dashboard will need to quote an appropriate access in order for the request to be successful.

Example of a obtain PeIs request carrying an RPT:

```
GET/IiwiaWF0IjoxNTE2MjM5MD?user=owner HTTP/1.1
Host: www.CA.ObtainPeIs.com
Authorization: Bearer
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI
6IkpvaG4gRG9lIiwiaWF0IjoxNTE2MjM5MDIyfQ.cThIIoDvwdueQB468K5xDc5633se
EFoqwxjF_xSJyQQ.
```

10.6 Response

If the obtain PeI request has been determined to be authorised (as a result of introspecting the RPT or the cached result of previous introspection) then the consent and authorisation service will respond back to the dashboard with the user's PeI(s).

```
HTTP/1.1 200 OK
Content-Type: application/json;charset=UTF-8
...
{
"peiList": ["pei:aJ001:WGF45920EJH348ASEWQ0284",
            "pei:sW001:WGF345560EJH348OGIIO4682",
            "pei:jk001:WGF45920EJH348ASEWQ0284"]
}
```

10.7 Error handling

If the obtain PeI request is invalid the consent and authorisation service will respond back to the dashboard with a HTTP 401 error code and an appropriate error message.

If the request is missing an RPT or has an invalid RPT, then the consent and authorisation service in its response will provide a WWW-Authenticate header with the authentication scheme UMA, with the issuer URI from the authorisation server's discovery document in an as_uri parameter indicating the URL of the authorisation server, where the dashboard should reach for further interactions to get an access token and the permission ticket in a ticket parameter. This will enable the dashboard to be able to initiate the authorisation protocol and obtain a valid RPT with the authorisation server.

For example:

```
HTTP/1.1 401 Unauthorized
WWW-Authenticate: UMA realm="PensionDashboard",
  as_uri="https://as.pdp.com",



ticket="eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiw
ibmFtZSI6IkpvaG4gRG9lIiwiaWF0IjoxNTE2MjM5MDIyfQ.cThIIoDvwdueQB468K5
xDc5633seEFoqwxjF_xSJyQQ"
...
```

If, after attempting authorisation as above, a dashboard cannot obtain an access token (RPT) it should stop attempting to do so, perhaps informing its user of remedial action. This condition is possible if the user has withdrawn consent for that dashboard, or in some other way consent to share PeIs is not current.

11 Obtain PeI configuration API

11.1 Summary of obtain PeI configuration API

Dashboards will need to dereference the pension identifier, which will be in the form of a URI in order to compose the URL to make the HTTP GET request on to view pension details. Dereferencing a PeI means taking the PeI which is a URI (format 'pei':<holdername>:<assetidentifer> eg"pei:aed123aq:WGF45920EJH348ASEWQ0284") and breaking it into components, looking up the holdername, eg 'aed123aq' in a configuration table to derive a scheme name, eg 'aviva.dashboard/pei', and composing a URL, eg https://aviva.dashboard/pei/WGF45920EJH348ASEWQ0284.

The dereferencing configuration table at the dashboard is based on master data maintained from the governance register. The master will be accessible via an API by registered dashboard providers.

11.2 Hosting

The API will be hosted on the governance register.

11.3 Format

The API will be a REST API using JSON encoded as UTF-8.

11.4 Authorisation

The dashboard client will authenticate itself with the authorisation server at run time as defined in rfc8705 section 2 . In addition to this, all end point connections are secured using mutual TLS.

11.5 HTTP method

The dashboard will make a HTTP GET request to the obtain PeI configuration endpoint with the parameters containing the variable name (ie pei) and corresponding value (ie holdername) – ie the dashboard attempts to GET the URL https://GR.ObtainPeIsConfig?pei”=aj001,sw002,...” this will be for a specific set of PeIs. The dashboard can alternative request all of the PeI configuration from the governance register by using the appropriate query parameter in the HTTP GET request – ie the dashboard attempts to GET the URL https://GR.ObtainPeIsConfig?pei=all.

11.6 Response

The governance register will respond with the corresponding hostname for each of holdername.

HTTP/1.1 200 OK
Content-Type: application/json;charset=UTF-8
...

```
{
"av001": "aviva.dashboard/pei",
"sw002": "scottishwidows.dashboard/pei"
{
```

11.7 Error handling

If the request sent by the dashboard fails then governance register is expected to respond with the appropriate HTTP status code and error message.

400 Bad Request

Potential bad requests examples:

- *dashboard is not sending a HTTP GET*
- *parsing error by governance register*
- *schema not configured correctly*

503 Service Unavailable

- *service is down for maintenance or overloaded by requests*

12 Dashboard redirection protocols

Dashboards expose no interfaces (other than their redirection endpoint, which is used to unwind a previous redirection to the relevant consent and authorisation (C&A) redirection interface).

Redirection from dashboard to the C&A is a vital constituent of these processes and for dashboard designers it is vital to understand in connection with the strict 'APIs'.

Dashboards must redirect their user agents to either of C&A's interfaces – UMAGrant.ClaimsRedirection or ConsentandControl.Redirect. The ClaimsRedirection interface is part of the UMA authorisation flow. The Consent.Redirect interface handles requests of type:

- *find (including pull Peis)*
- *refresh PAT (RS failure case)*
- *consent*
- *account deletion (GDPR)*

For every interaction made by the dashboard to the consent and authorisation service, it has to mint a new RQP token.

13 Client registration

Registration of OAuth clients (ie dashboards and data providers) is required for the operation of the UMA profiles. The governance register (which includes the UMA authorisation server as the software entity managing software client registration) will provide services for client registration.

Static registration may be more secure but dynamic registration may be more expected by industry participants, may involve fewer manual processes, and is necessary to support public client types (especially non-confidential ie SPAs, native apps).

In order for PDP to offer dynamic client registration, it requires the authorisation server (AS) to support dynamic registration [ODynClient] and to provision clients appropriately. The AS needs to support software statements to bootstrap the registration process securely.

Client authentication to the AS will be defined in accord with dynamic registration requirements.

The alpha phase of the programme will focus on static client registration. Dynamic client registration will be looked at once the requirements to support this have been refined with the supplier.

14 Appendix

14.1 Glossary

Term	Definition
resource owner (ie pension owner)	The resource owner is a user or legal entity that is capable of granting access to a protected resource
client (ie dashboard)	The client is an application that is capable of making requests with the resource owner's authorisation and on the requesting party's behalf
UMA resource server (ie data provider)	The resource server hosts resources on a resource owner's behalf and is capable of accepting and responding to requests for protected resources
UMA authorisation server	Part of the consent and authorisation service – the authorisation server protects resources hosted on a resource server on behalf of resource owners. It manages and applies the resources owner's policy
PMT – permission token	Gives permission for a dashboard to initiate the authorisation protocol in order to authorise a retrieval request
RQP – requesting party	Identifies the current user and their role in session at the dashboard when requesting access. Can be either a pension owner or a delegate
RPT – requesting party access token	Needed by the dashboard to authorise its view call to the data provider endpoint in order to retrieve pensions details related to the PeI. Each PeI will have a unique RPT (ie one to one association)
PCT – persistent claims token	This claim binds the asserted user at dashboard and role to that user's assured identity at the ecosystem and, where applicable, professional status. In doing so it acts as a medium-term authenticator correlator, helping reduce user friction in subsequent sessions, when attempting to retrieve pension details
PAT – protection API token	OAuth2 token scope UMA protection which represents the resource owner's authorisation for the resource server to manage federated authorisation at the authorisation server (permits APIs to register, obtain PMT, introspect RPTs)
user account token	Oauth2 authorisation grant issued by the authorisation server to the resource server, so that it can be exchanged at the authorisation server's token endpoint for a PAT