Pensions Dashboards Programme

# Technical standards

July 2022

pensionsdashboardsprogramme.org.uk

Money & Pensions Service

# Contents

# 1. Introduction

## 1.1 Purpose

This document outlines the technical standards for pensions dashboards that need to be adopted by industry participants: qualifying pensions dashboard services (QPDS); and pension providers (trustees and manager of occupational pension schemes as well as the managers of stakeholder and personal pension schemes). References to data providers includes pension providers and third parties who are supporting them comply with their dashboard duties. It also contains important guidance and background QPDS and data providers should familiarise themselves with before they comply with the standards.

Technical standards are separate from, but designed to complement, the Financial Conduct Authority's (FCA) regulatory framework. As the FCA regulates the conduct of firms carrying out an activity, the FCA's Handbook rules will apply to QPDS firms and can impose standards on those firms (aligned to FCA's statutory objectives) when carrying out the qualifying pensions dashboard service. The FCA will consult on its proposed Handbook rules in due course.

## 1.2 Scope

The following core areas are covered in the document:

- part 1: technical overview: sections 2 to 3
- part 2:
    - API standards: sections 4 to 13
    - technical standards: section 16
- glossary: section 17

Part one provides a technical overview of how the ecosystem operates. This is essential background reading for participants to be able under to understand the mandatory API and other technical standards detailed in part two.

# 2. Technical overview

## 2.1 Client registration

Registration of OAuth clients (ie QPDS 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.

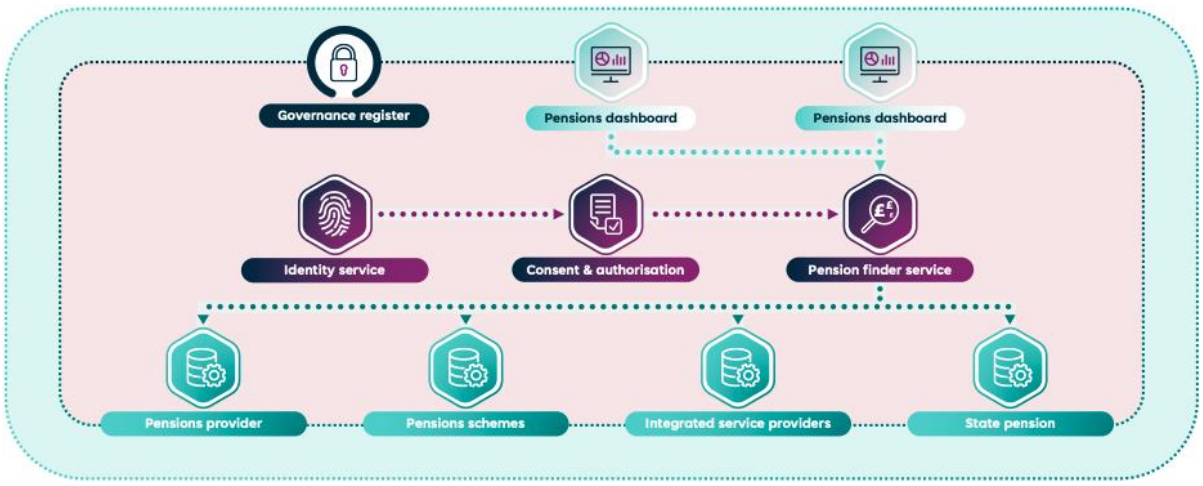
Static registration may be more secure but dynamic registration may be expected by industry participants, may involve fewer manual processes, and is necessary to support public client types (especially non-confidential ie SPAs, native apps).

In order for dynamic client registration to be effective it requires the AS to support dynamic registration [ODynClient] and to provision clients appropriately. The AS must support software statements to bootstrap the registration process securely.

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

Currently static client registration is being used. In future, dynamic client registration will be provided and this document updated accordingly.

**Overview of the pensions dashboards ecosystem primary components primary components**
This diagram shows each component involved in the protocol interactions in this document.



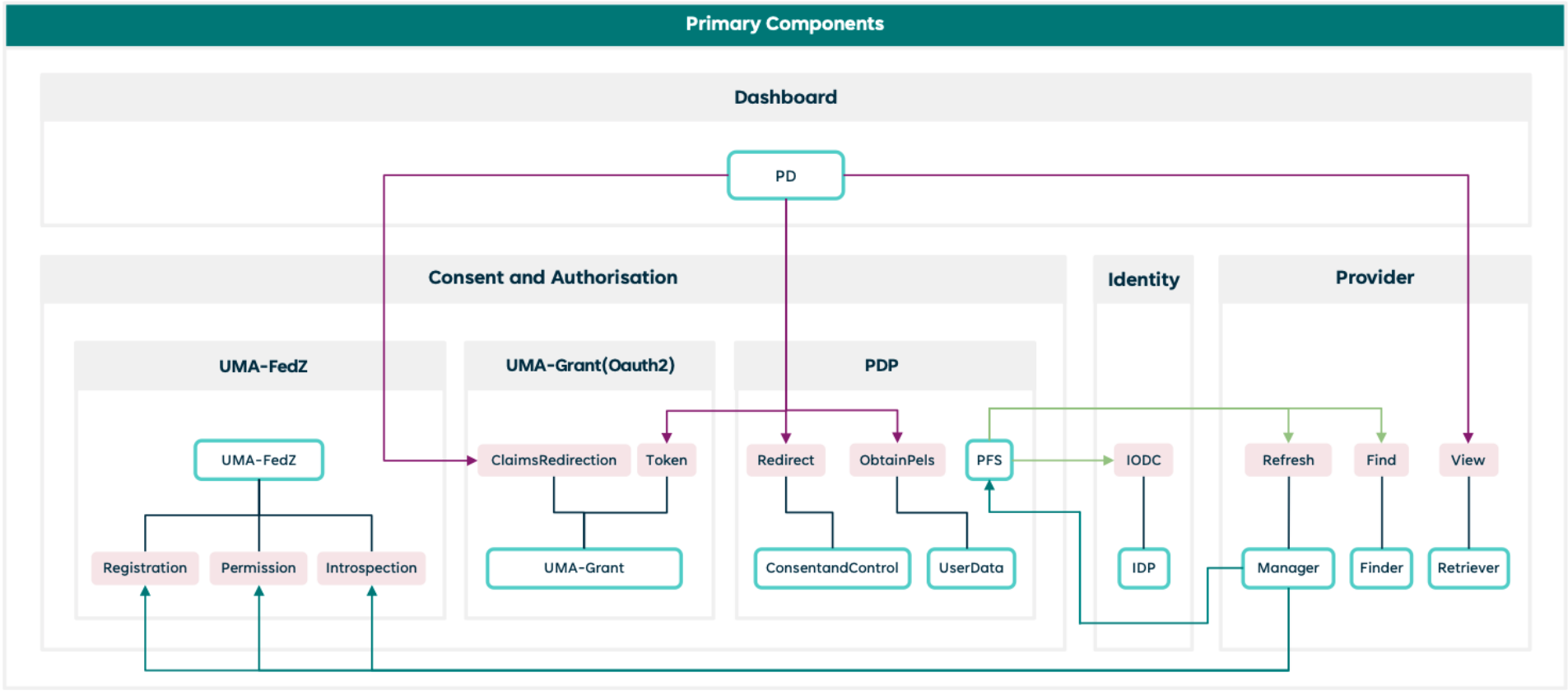

## 2.2 Pension dashboards ecosystem primary components

Components are presented in packages and each package corresponds to a participant in the ecosystem: dashboard, consent and authorisation (C&A) service, data provider, identity service. Within a package the components expose interfaces (blue lines) and call interfaces of other components (lines of other colours dependent upon the caller) eg the data provider interfaces are called by dashboards (red) and by C&A components (green).

Calls within packages are not shown on this diagram eg PDP.ConsentandControl – handling redirections – deeply interacts with UserData and the UMA authorisation components eg PDP.ConsentandControl interacts with PFS when it initiates a search, passing consents and search parameters and Authorisation Server grants.

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

PDs 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', 'consent' and 'refresh'.


Primary Components
Dashboard
PD
Consent and Authorisation
Identity
Provider
UMA-FedZ
UMA-Grant(Oauth2)
PDP
UMA-FedZ
ClaimsRedirection
Token
Redirect
ObtainPels
PFS
IODC
Refresh
Find
View
Registration
Permission
Introspection
UMA-Grant
ConsentandControl
UserData
IDP
Manager
Finder
Retriever

The authorisation server is not explicitly shown on the diagram: it is part of the consent and authorisation package and comprised of the UMA components and *some* of the functions of consent and control and user data.

The **two UMA2 interfaces** can be seen: federated authorisation is called by a data provider, and UMA Grant by the dashboard.

Note that, since an UMA authorisation server is a specialisation of an OAuth2 authorisation server, the Token interface, part of UMA-Grant(OAuth2), is *also* the same endpoint called by the dataata provider when it needs to obtain a PAT (Protection API Access Token), which is an OAuth2 access token with scope uma_protection, subsequently used against the UMA Federated Authorisation Protection API (UMAFedz).

Apart from UMA support itself, the ecosystem package shows the **other primary components of the C&A** service. The user-facing consent services manage user interaction and policy. User data contains the policy and registered PeIs for each user; it exposes the interface for dashboards to obtain PeIs for their users. The consent and control component also manages user and authorisation processes and repositories (user data , selectable names for user directed find, etc). PFS performs search orchestration after the consent and control component has established the context, which includes proving user identity using the identity service, establishing user data , and other tasks as necessary.

The i**dentity service** interface (IDP.OIDC) is simplified in this diagram (it is also a specialisation of the standard OAuth2 Authorisation Service endpoints).

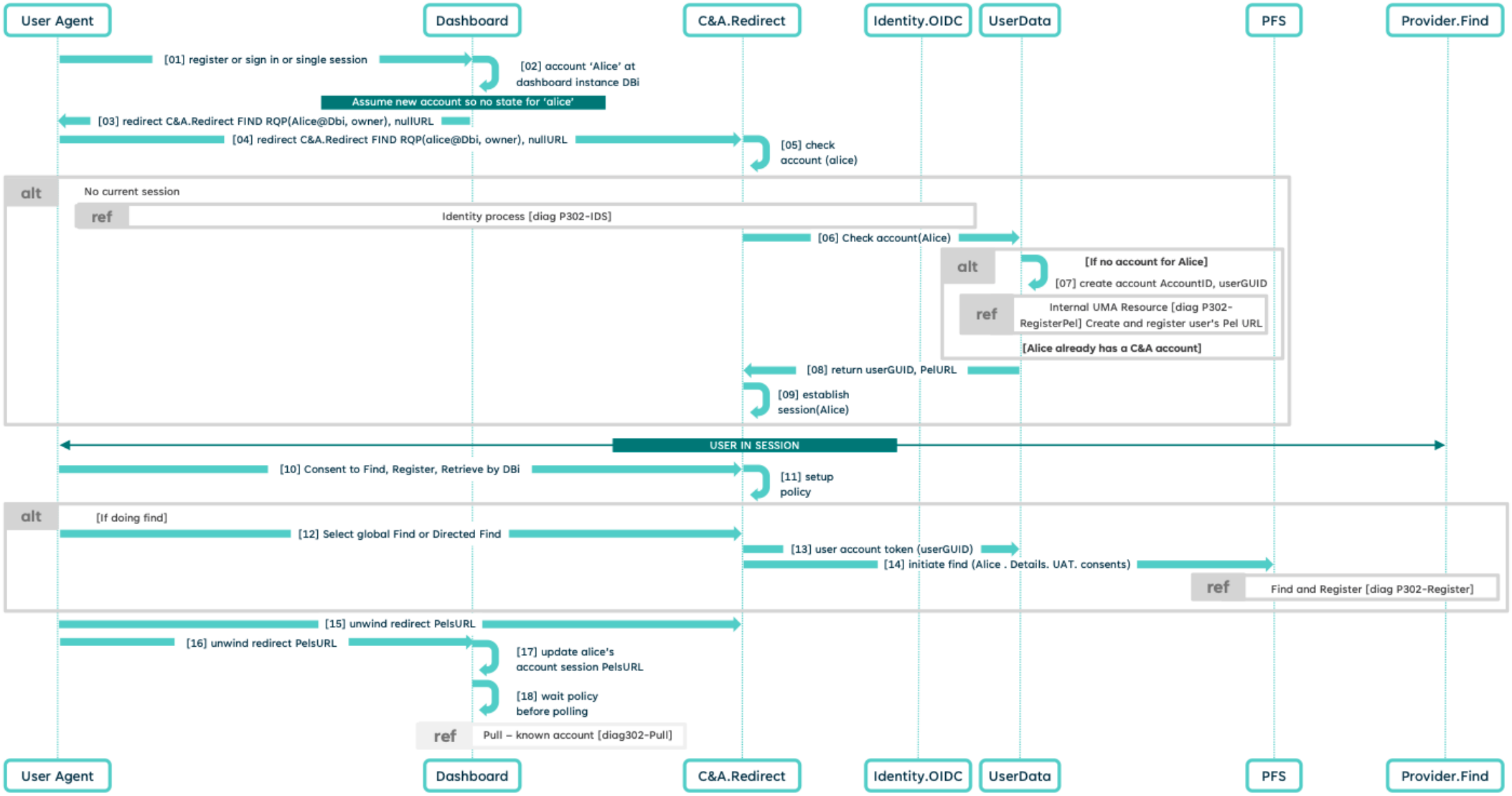
# 3. Sequence flows

## 3.1 Initial find

This section presents a flow for a new user at a dashboard, initiating a find activity at the C&A.

In this case the dashboard knows it has a new user (unless it is a stateless dashboard when it *assumes* it has a new user) and redirects to the C&A to process a find operation. The condition around Step 12ff is discussed in a later section on obtaining PeIs URL from dashboards. In the simple 'new find' variant, these find steps will be performed, specifically the PFS will be invoked, Step 14, to orchestrate calls to several/all data providers

# Initial find



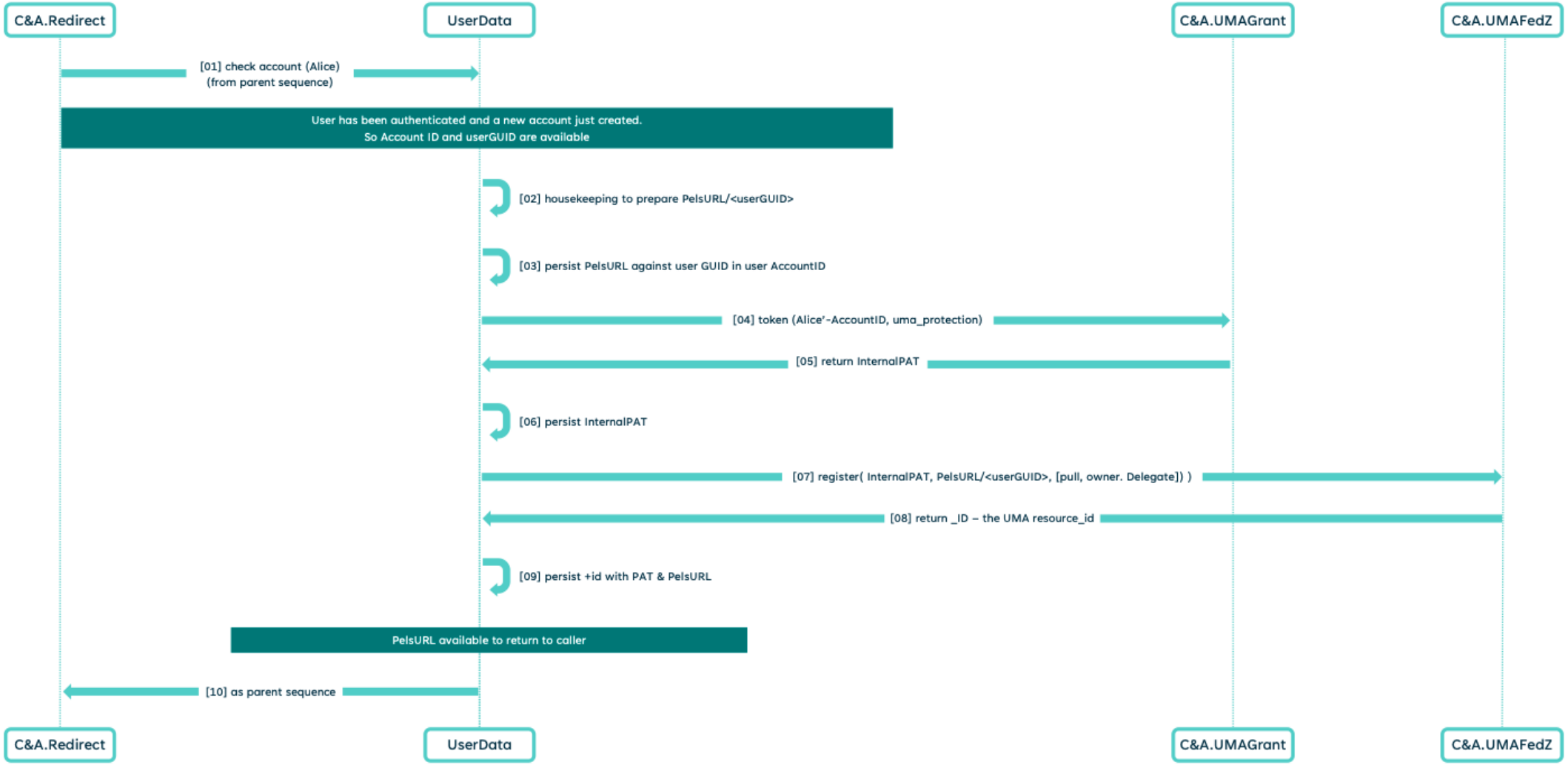
### *3.2 Internal register obtain PeIs-URL*

This sequence is nested in the above flow. After creating a C&A account at step 07 this sequence registers the protected resource for the owner user's PullPeIs resource. In this case both the UMA authorisation server and the UMA Resource Server which presents the PullPeIs resources are in the *same security domain*, ie are part of the C&A service. Accordingly, there is the possibility of significant optimisation (customisation) of the relationship between the AS and the PullPeIs Resource Server, yet to still offer the same external UMA2 grant based access to the protected resource by dashboards.

The following takes an 'UMA2 Federated Authorisation' approach.

In Step 04 the parameter Alice'-AccountID is Alice's identity token, available from the parent sequence, along with the new derived AccountID, from which a token can be constructed in the trusted UserData RS. This token is used as an authorisation grant for the authorisation server to issue an internal PAT. In implementation, this could use the OAuth2 JWT-bearer grant. Since the AS and the RS (the UserData component) are coupled and in the same domain, the AS

# Obtain Pels URL



could check that Alice's account (AccountID) exists (not shown on the diagram) or simply trust the internal caller and signature, and issue the requested access token (the InternalPAT) at step 05.

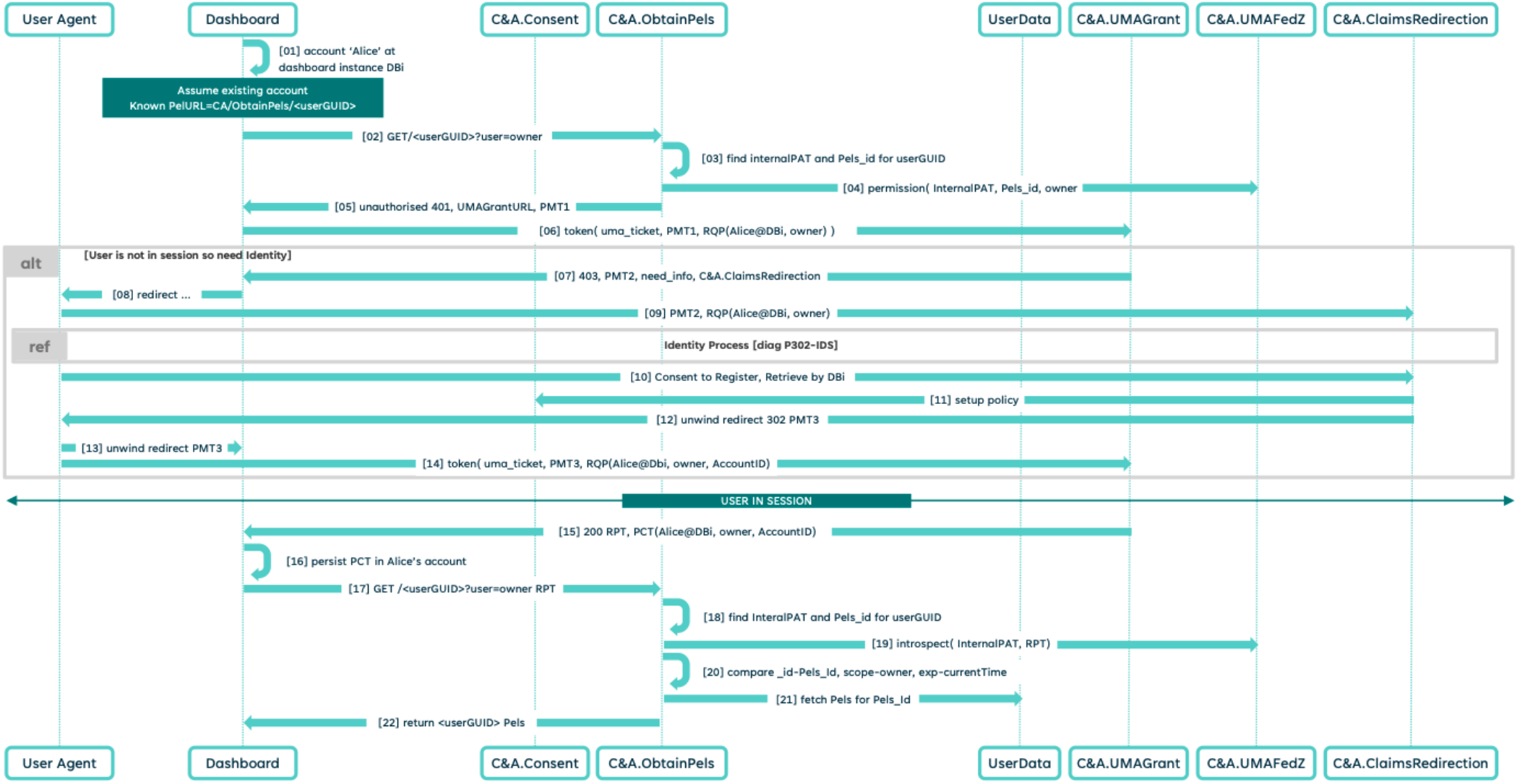
## 3.3 Pull PeIs

Once the dashboard knows the user's C&A account in the form of the relevant protected PeIs resource, it can pull the PeIs as follows. The dashboard attempts to GET the URL https::://CA.ObtainPeIs/<userGUID> with a parameter of the user is 'owner'. Initially, this will fail because the dashboard does not quote an appropriate access token, but it will trigger the UMA authorisation dance which obtains the RPT which then succeeds in the subsequent call.

This flow is not limited to being a direct successor to the initial find flow above, it also stands alone and can be performed whenever necessary

- immediately after find new
- for up to the defined SLA after an initial find as the (slower) data providers find pension records
- after repeat finds (user-controlled timing, user directed finds)
- when a delegate (financial adviser or guidance officer) is initialising their records for the client in which they have the client-specific PeIsURL in their communication from the C&A server
- when an owner-user initialises a new dashboard by importing PeI data from a previous one, the UMA PeIs resource would bootstrap the process of initialising the new dashboard. New pensions dashboard attempting to obtain the PeIs would initiate the whole of this process, including new identity assertion, account update, new consent and policy for the new dashboard)

# Pulls Pels


User Agent
Dashboard
C&A.Consent
C&A.ObtainPels
UserData
C&A.UMAGrant
C&A.UMAFedZ
C&A.ClaimsRedirection
[01] account 'Alice' at dashboard instance DBi
Assume existing account
Known PelURL=CA/ObtainPels/<userGUID>
[02] GET/<userGUID>?user=owner
[03] find internalPAT and Pels_id for userGUID
[04] permission( InternalPAT, Pels_id, owner
[05] unauthorised 401, UMAGrantURL, PMT1
[06] token( uma_ticket, PMT1, RQP(Alice@DBi, owner) )
alt
[User is not in session so need Identity]
[07] 403, PMT2, need_info, C&A.ClaimsRedirection
[08] redirect ...
[09] PMT2, RQP(Alice@DBi, owner)
ref
Identity Process [diag P302-IDS]
[10] Consent to Register, Retrieve by DBi
[11] setup policy
[12] unwind redirect 302 PMT3
[13] unwind redirect PMT3
[14] token( uma_ticket, PMT3, RQP(Alice@Dbi, owner, AccountID)
USER IN SESSION
[15] 200 RPT, PCT(Alice@DBi, owner, AccountID)
[16] persist PCT in Alice's account
[17] GET /<userGUID>?user=owner RPT
[18] find InteralPAT and Pels_id for userGUID
[19] introspect( InternalPAT, RPT)
[20] compare _id-Pels_Id, scope-owner, exp-currentTime
[21] fetch Pels for Pels_Id
[22] return <userGUID> Pels
User Agent
Dashboard
C&A.Consent
C&A.ObtainPels
UserData
C&A.UMAGrant
C&A.UMAFedZ
C&A.ClaimsRedirection

### 3.4 Provider registers PeIs

This flow is referenced from initial find flow above.

The PFS's only role is to orchestrate the calls, step 01 in the diagram below, the receipt of each 'find' call should be acknowledged using low level http 202 code; note this has nothing to do with whether the find actually results in the location of a pension.

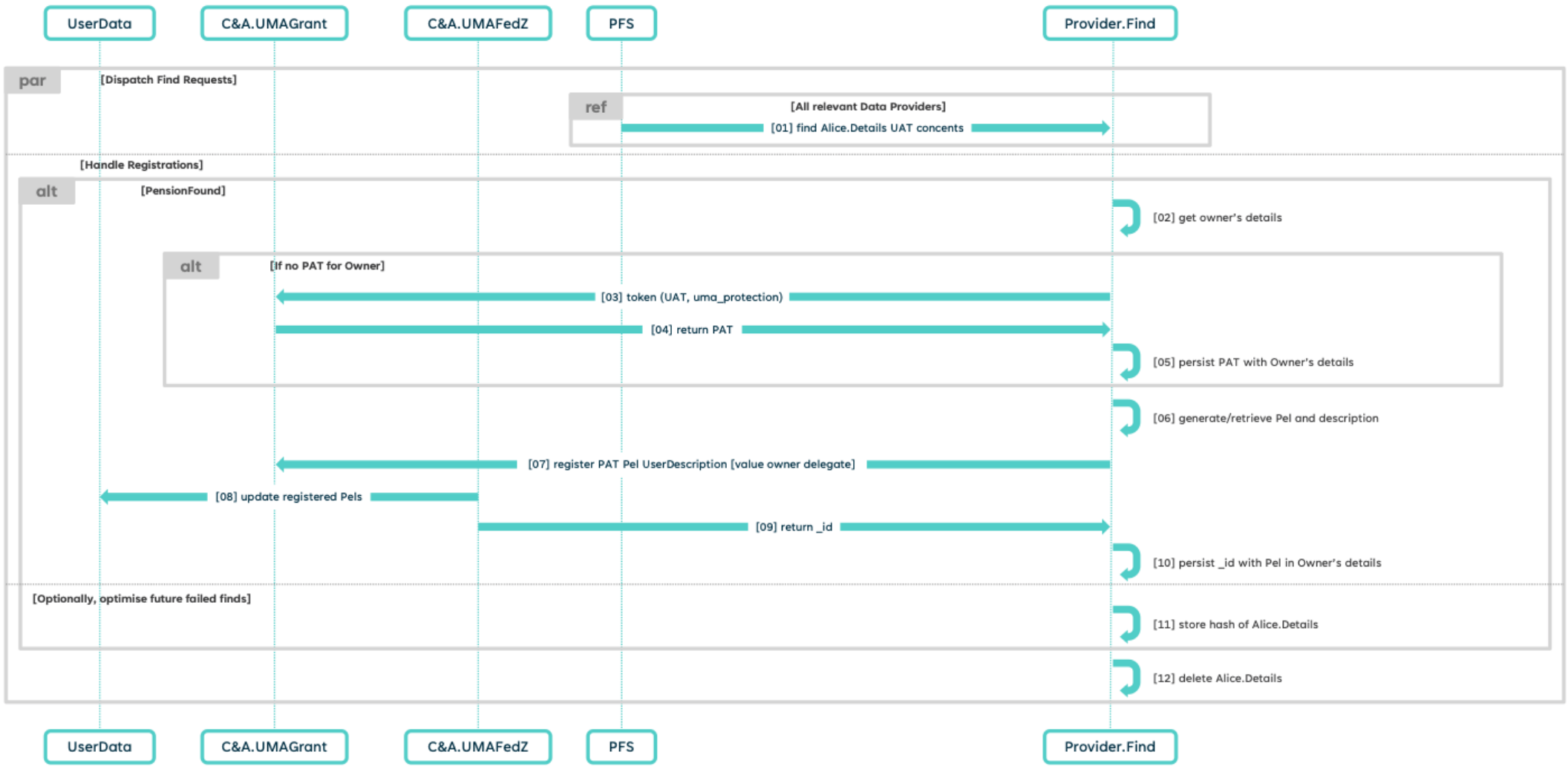
When a pension is found it is registered at the C&A service by the data provider using the UMA Protection API and an access token issued for the purpose, a PAT.

If no pension is located (step 11) the data provider may store a hash of the some of the details used in the search so that when a subsequent search is initiated by the same user the provider does not waste resources in running a repeat search.

Consents in this flow are assumed to permit find and register. If register is not permitted then the data provider could still contact its customer using its existing details as it knows that the customer is interested in their pension (but this is a business decision for the provider).

If a pension is located (and consents permit) the provider should register the corresponding PeI with the C&A server as per the flow (steps 02ff).

# Register Pels



Note that Alice's personal details originated from the IDP/C&A service are never retained by the data provider whether or not the find is successful, step 12.

The case where the find failed has been mentioned above. However, in cases where the find succeeded (so steps 02-10 apply) a data provider will need to arrange for **step 02 'get Owner's Details'** to function correctly, for view and for find, whether or not the same user has executed a find operation in the past. The pensions dashboards ecosystem requires that these details are:

- any identifier the provider wants to use for the owner's account
- any internal keys or similar which enable the provider to locate relevant internal records associated with the owner and their assets
- the user's authorisation server (Of course in the PD ecosystem at least initially this will be the one C&A Server, but theoretically this might change over time if users appoint other 'open finance' components, so the actual C&A AS URL should be kept in the RS from the outset; in any case it is required to be returned to the dashboard in the UMA protocol.)
- PAT when one exists
- associated PeIs and their descriptions, including a map of inbound URLs to the relevant PeI
- UMA _id for each PeI

Data providers will need to consider how to retrieve this data in two cases:

- on the basis of the inbound View URL. This is a map of URL->PeI->_id and from that to the owner & PAT
- on the basis of subsequent 'find' requests for the *same* user. The provider needs to decide whether to repeat the find operation itself, which should result in the same internal owner identifier and internal keys (and thus find the above details), or keep a hash of the inbound personal details and use that to index the above details.

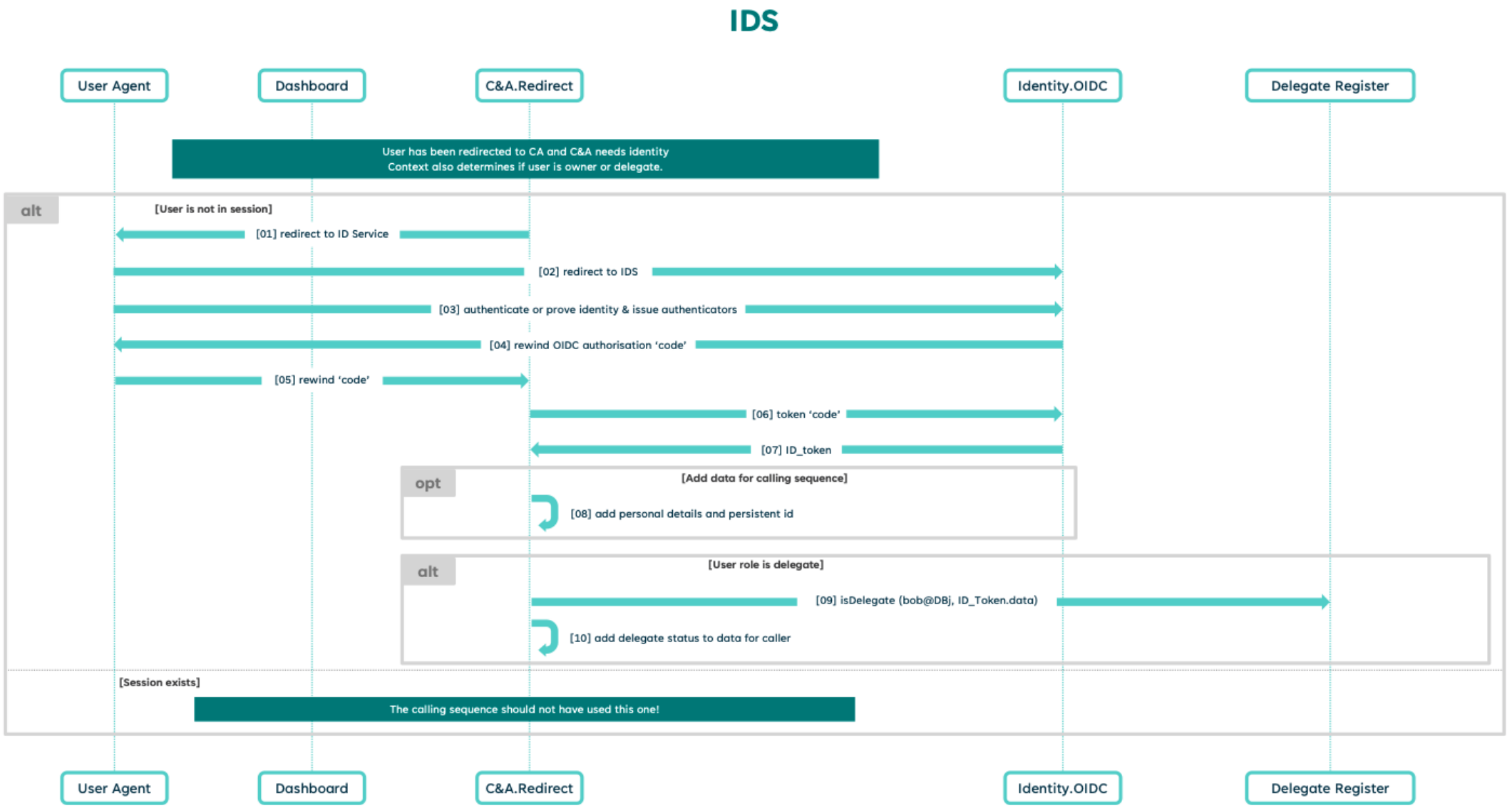
**Step 03** shows the use of a temporary credential as an **OAuth2 grant** of type JWT-bearer, as described in section 3.5.

**Step 06 'generate or retrieve PeI'** also indicates a decision for providers. A PeI is a unique URI (not URL) identifying a 'asset' / 'pot' / 'pension' uniquely and persistently. It is *independent* of the owner of the asset. Thus it can be allocated *at any time* from the creation of the asset until its initial registration with the PD C&A service. There are choices open to data providers in their business processes and in their preparation for PD integration based on the properties of the PeI, its behaviour across providers, and the provider's decisions on processing 'maybe' or 'uncertain finds'. These matters need to be discussed with industry representatives and PDP should issue advice or mandation.

## 3.5 Identity process

Whenever the C&A service needs to determine a user's identity it uses the identity service, which is (assumed to be) a federation of external identity providers, interfaced via some form of hub. The purpose of this section is to present how this service integrates with the C&A.

There are two key use cases, dependent upon the 'role' of user. The user may be a pension *owner* or the user may be a *delegate* of one or more owners. Both user roles need identity proofing. PDP has not yet determined the Level of Confidence (according to GPG45) required for proofing of each role. The identity service will issue authenticators to each user to a certain standard (GPG44) to enable reauthentication without re-proofing. In addition, delegates will have to have their status as a valid delegate checked. This check will probably be recorded in the governance register which will surface an API to be used for verification in the following flow.


IDS
User Agent
Dashboard
C&A.Redirect
Identity.OIDC
Delegate Register
User has been redirected to CA and C&A needs identity
Context also determines if user is owner or delegate.
alt
[User is not in session]
[01] redirect to ID Service
[02] redirect to IDS
[03] authenticate or prove identity & issue authenticators
[04] rewind OIDC authorisation 'code'
[05] rewind 'code'
[06] token 'code'
[07] ID_token
opt
[Add data for calling sequence]
[08] add personal details and persistent id
alt
[User role is delegate]
[09] isDelegate (bob@DBj, ID_Token.data)
[10] add delegate status to data for caller
[Session exists]
The calling sequence should not have used this one!
User Agent
Dashboard
C&A.Redirect
Identity.OIDC
Delegate Register

The implementation of the standard OIDC identity service will comprise of an authorisation endpoint (handling redirection of the user agent to so the user can authenticate and prove their identity) and a token endpoint from which the identity assertion (ID Token) will be retrieved, using OIDC authorisation code flow. OIDC Core also defines a 'user_info' endpoint to obtain additional claims, in addition to those which may be in the ID Token, if the pensions dashboard identity profile or standards requires it. In addition, it is likely that the OIDC interface will be brokered via a hub component (not shown in this document) so that a federation of identity providers can be supported.

The context of this sequence (eg initial find above) will determine details of the information required from the Identity process. For example, the call will determine whether the user is a delegate on a delegate dashboard (eg bob@DBj) or an owner user. The sequence should only be called when the caller has checked there is no current session for the owner user. After this flow the caller may also create the C&A account for the owner user and establish the required session as appropriate, using the temporary data noted in the flow.

The dashboard may have already authenticated the user using the ecosystem identity service (see p301 6.1 for discussion). The fact that the user is authenticated by an IDP may be added to the redirect parameters (probably adding an element to the RQP so an RQP is comprised of: alice@DBi, owner, and idp=<IDP>). Given that the Pensions Dashboards Programme mandate that the IDPs in its federation must maintain an open session (for a controlled period) then the redirection to the identity service in the above flow can immediately return the OIDC grant code without requiring the user to reauthenticate at the Open ID provider (IDP).

## 3.6 View

When a dashboard has obtained PeIs for its user, it will usually seek to view the pension details, by GETing the *dereferenced* PeI. This action will initially fail, as there is no access token, initiating the UMA2 Grant authorisation dance.

**Dereferencing** a PeI means taking the PeI which is a URI (format 'pei':<holdername GUID>:<asset GUID> e.g. "pei: 85624cd0-de31-455c-8ce8-f26515a29577: 9112820e-7d9d-47e0-bbfe-129b8afbab0d") and breaking it into components, looking up the holdername, e.g. '85624cd0-de31-455c-8ce8-f26515a29577' in a configuration table to derive a host name, eg 'aviva99.dashboard/pei', and composing a URL eg https://aviva99.dashboard/pei/85624cd0-de31-455c-8ce8-f26515a29577 . It is this URL which is an UMA protected resource; the URL of the pension details to be accessed by an http GET with appended role parameter, eg '?user=owner'.

The governance register API will map the <holdername> element of the PeI to the current URL of the provider's view endpoint (example above). A cache entry might be deleted if a resulting hostname is not resolved or periodically based on pensions dashboards standards.

**Dashboard preparation** a user may select one or more descriptions of their pensions to view (Step 01). The dashboard finds related control information whether this be from a persistent store (for dashboards which have accounts & appropriate security controls) or from its current session. For each user description of an asset there will be a PeI and may be an existing access token, RPT. If the user has been authenticated to the identity service recently from that dashboard there may be a PCT available

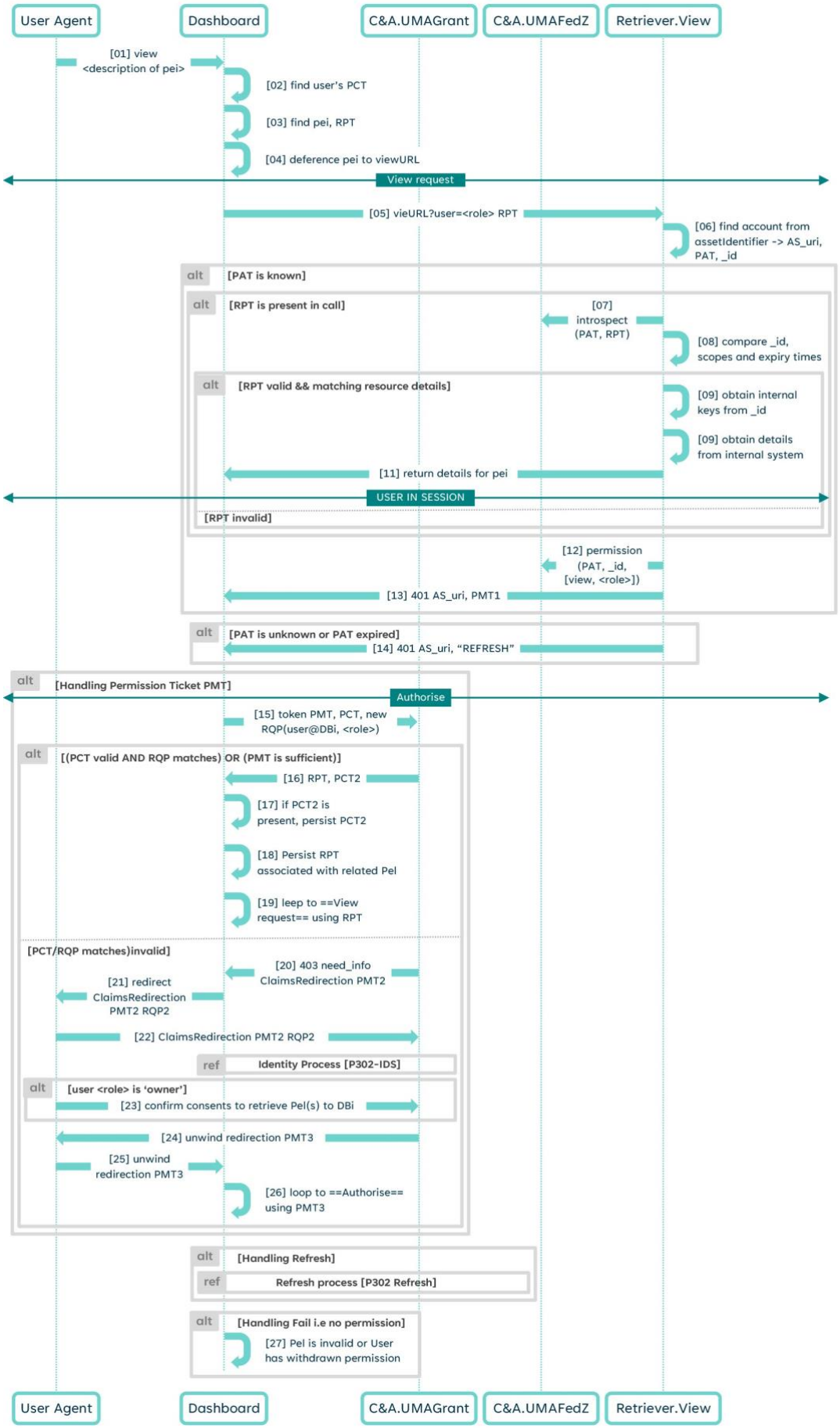
from that user's account at the dashboard. The PeI is dereferenced as discussed above, deriving the protected resource for the asset.

**Authorised view access** the 'happy path' in which the provider locates the UMA control information based on analysis of the URL (step06) and there is a valid access token (07) which matches the user's role (owner or delegate) and other information (08), results in the api returning the pension details associated with the PeI to the dashboard.

There are three types of **authorisation failure**:

- those for which the AS can provide an UMA permission token (PMT) to the RS which is returned to the dashboard
- the case in which the PAT has expired, so the UMA AS cannot respond to the provider's RS request for a permission ticket. This is handled by a PDP-specific error message requesting 'REFRESH' processing which is discussed in a later section
- final failure in which the RS or AS does not return a PMT nor a specific instruction. In this case the dashboard must assume that the PeI is invalid or that the user has withdrawn consent for that dashboard (in spite of having been directed to the C&A as part of the above automatic handling processes)

# View


User Agent
Dashboard
C&A.UMAGrant
C&A.UMAFedZ
Retriever.View
[01] view <description of pei>
[02] find user's PCT
[03] find pei, RPT
[04] deference pei to viewURL
View request
[05] vieURL?user=<role> RPT
[06] find account from assetIdentifier -> AS_uri, PAT, _id
alt [PAT is known]
alt [RPT is present in call]
[07] introspect (PAT, RPT)
[08] compare _id, scopes and expiry times
alt [RPT valid && matching resource details]
[09] obtain internal keys from _id
[09] obtain details from internal system
[11] return details for pei
USER IN SESSION
[RPT invalid]
[12] permission (PAT, _id, [view, <role>])
[13] 401 AS_uri, PMT1
alt [PAT is unknown or PAT expired]
[14] 401 AS_uri, "REFRESH"
alt [Handling Permission Ticket PMT]
Authorise
[15] token PMT, PCT, new RQP(user@DBi, <role>)
alt [(PCT valid AND RQP matches) OR (PMT is sufficient)]
[16] RPT, PCT2
[17] if PCT2 is present, persist PCT2
[18] Persist RPT associated with related PeI
[19] leep to ==View request== using RPT
[PCT/RQP matches)invalid]
[20] 403 need_info ClaimsRedirection PMT2
[21] redirect ClaimsRedirection PMT2 RQP2
[22] ClaimsRedirection PMT2 RQP2
ref Identity Process [P302-IDS]
alt [user <role> is 'owner']
[23] confirm consents to retrieve PeI(s) to DBi
[24] unwind redirection PMT3
[25] unwind redirection PMT3
[26] loop to ==Authorise== using PMT3
alt [Handling Refresh]
ref Refresh process [P302 Refresh]
alt [Handling Fail i.e no permission]
[27] PeI is invalid or User has withdrawn permission
User Agent
Dashboard
C&A.UMAGrant
C&A.UMAFedZ
Retriever.View

The **authorisation process** (Step 15) uses the current PMT (always present), and current PCT (if there is one) and a new RQP token, minted by the dashboard for every relevant call to the C&A service. The RQP (Requesting Party token) is an assertion by the dashboard that a specific user known to it (eg 'alice') is present at the interface of a specific dashboard instance (e.g. DBi), and that the user is acting in a specific role (i.e. owner or delegate).

The AS makes the authorisation determination based on the state (contained in the PMT at step 15 and in the AS) and the other parameters. If the authorisation request is successful, the AS returns an access token for the specific protected resource (RPT) and it may issue a new or replacement PCT. The PCT (Persistent Claims Token) binds the user at the dashboard to the C&A's view of the identity (derived from the external identity service) and the user's validated role. When an authorisation request is successful the dashboard should retry the view request.

If the AS determines that it cannot grant access, but that it can continue the authorisation process, it issues another PMT and requests that the client (dashboard) redirects the user to its Claims endpoint for an interactive session with the user. There the appropriate identity uplift occurs and if the user is an 'owner' the appropriate consents are refreshed. (Details of interaction between the claims redirection code and the Consent and UserData components are omitted here, but see also the sequences Initial find and pull PeIs above.) The AS also needs to consider whether the policy of the owner for a specific PeI and specific delegates matches the recently uplifted user. If identity uplift was successful and policy matches PeI and person, the AS issues another PMT and unwinds the redirection back to the dashboard. The dashboard uses the new PMT to re-attempt the authorisation call.

## 3.7 Refresh process

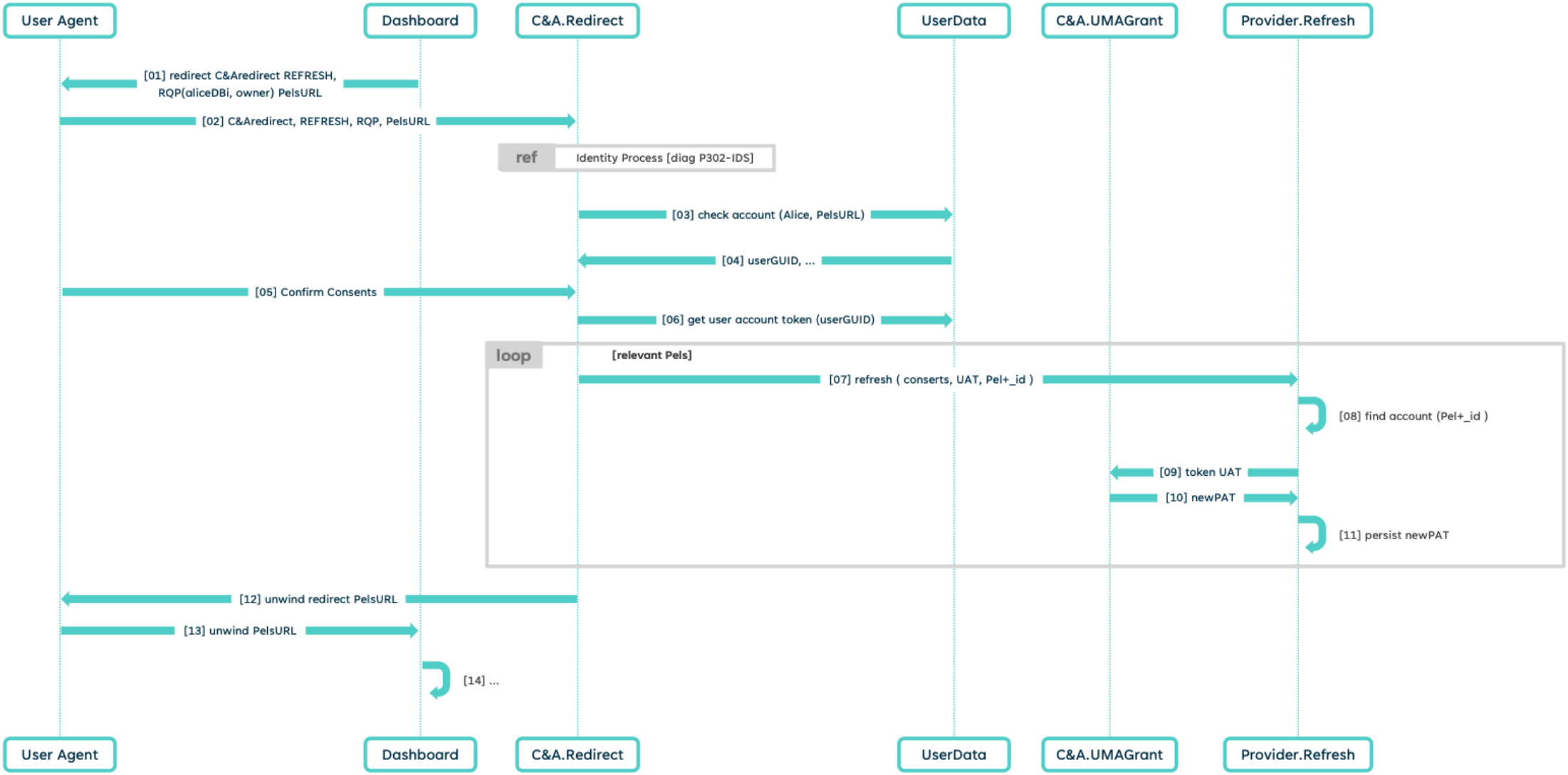
Only owner-users can perform refresh (not delegates); same is true of the other redirect purposes – find and consent.

Refresh processing may be entered as a result of failure of a view attempt in which a PAT has expired, or in *routine* use of the C&A Consent service by the user (eg in adding dashboards, changing consents or stepping up identity after PCT expiry after 90 days eg user directed find or repeat find).

The owner-user is redirected to the C&A service so that consents can be confirmed or modified and, if necessary, the identity can be uplifted. If the user is in session at the C&A then there may be no reason to perform a reauthentication. (This is not shown on the diagram but simply skips the identity process.)

The flow shows the account details being checked, including the dashboard-provided user's PeIsURL. There are error conditions related to dashboard and user which can be corrected as part of this process (not shown) eg the user's correct PeIsURL is returned as described in the section 'Initial find' and the RQP can be used to check that the user has consent policy for that dashboard.

# Refresh


User Agent
Dashboard
C&A.Redirect
UserData
C&A.UMAGrant
Provider.Refresh
[01] redirect C&Aredirect REFRESH, RQP(aliceDBi, owner) PelsURL
[02] C&Aredirect, REFRESH, RQP, PelsURL
ref
Identity Process [diag P302-IDS]
[03] check account (Alice, PelsURL)
[04] userGUID, ...
[05] Confirm Consents
[06] get user account token (userGUID)
loop
[relevant Pels]
[07] refresh ( conserts, UAT, Pel+_id )
[08] find account (Pel+_id )
[09] token UAT
[10] newPAT
[11] persist newPAT
[12] unwind redirect PelsURL
[13] unwind PelsURL
[14] ...
User Agent
Dashboard
C&A.Redirect
UserData
C&A.UMAGrant
Provider.Refresh

Irrespective of the dashboard's stated purpose for the redirection (REFRESH, repeat FIND or CONSENT), or the purpose of the redirection to the Claims redirection endpoint during authentication (to seek periodic step-up), the C&A has the *opportunity* to run the loop to refresh PATs at the relevant providers of the PeIs of the user. Clearly if the redirection from a dashboard was for REFRESH, this implies that the dashboard had received an error from a provider during an attempted view, but even in this case the C&A needs to manage the process appropriately: for example if a user has withdrawn consent for any dashboard to view a PeI, or for registrations of PeIs from that provider, there is no need to attempt the refresh.

Note that, unlike the initial find process in which the owner-user's personal details are sent to perform a search at the provider, there is no need for these biographic details in performing a refresh. All that is needed are the parameters shown at step 07. The 'relevant PeIs' are those which a) are registered for the user's account, b) the RS hosting one or more of these PeIs is not subject to a current (user-directed, repeat) find[1] for the same user, c) the RS has not already been contacted[2] in the same refresh loop. For each relevant PeI, which has the relevant resource _id, the C&A can determine the provider RS and hence which Provider's Refresh end point to call.

The above flow assumes that the user does approve refreshing the PAT and that the dashboard which initiated the refresh is still consented to receive the owner's PeIs and other data. If not the return redirection (steps 13, 14) would not carry the current PeIsURL and may have a specific error code so the dashboard knows the user has withdrawn consent.

[1] Because if a find is happening for that user for the same RS, the RS will have a separate opportunity to refresh the PAT.
[2] Because only one PAT is needed for each user at an RS (not per PeI if the same user has more than one PeI).

# 4. API technical standards

## 4.1 Scope

The following areas will be covered in the within API technical standards section:

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

## 4.2 Transaction monitoring

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

The transaction identifier is issued by the party which initiates the transaction.

For transactions which PDP initiates, PDP will generate the identifier; for transactions which the data provider or QPDS initiates, the provider will generate the identifier.

## 4.3 Third Party Standards

These standards are built upon a number of third-party standards. Here is a list of them:

| Standard | Scope | Issuing Authority | Contact Details | Version |
|---|---|---|---|---|
| UMA Grant 2.0 | | Kantara Initiative | https://docs.kantarainitiative.org/uma/wg/rec-oauth-uma-grant-2.0.html | 2.0 |
| UMA Federated Authorisation 2.0 | | Kantara Initiative | https://docs.kantarainitiative.org/uma/wg/rec-oauth-uma-federated-authz-2.0.html | 2.0 |

# 5. Find API

## 5.1 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 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 PAT (protected API token) 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.

## 5.2 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.

## 5.3 Hosting

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

## 5.4 Format

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

## 5.5 Authorisation

This is a closed ecosystem with all end point connections secured using private PKI certificates issued by the governance register to suitably enrolled organisations. This enables connecting entities to establish a mutual TLS connection with the central infrastructure. For find requests only the PFS will be responsible for invoking the find API endpoints. Currently there are no additional API security requirements for the find API.

## 5.6 HTTP method

The PFS will be restricted to only make HTTP POST requests to each data provider find endpoint with the body of the request containing the find parameters as a signed JWTs.

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

| | |
|---|---|
| user_token | A combination of the following expressed as a JWT:<br><br>• verified identity details such as name, date of birth and postcode. These will have been selected by the C&A service from the verified details supplied by the Identity Service<br>• 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 (JSON web token) 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 |
| request_id | Transaction identifier GUID used for monitoring and reporting purposes |

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.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiaWF0IjoxNTE2MjM5MDIyfQ.SflKxwRJSMeKKF2QT4fwpMeJf36POk6yJV_adQssw5c",

"user_account_token":"eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiaWF0IjoxNTE2MjM5MDIyfQ.SflKxwRJSMeKKF2QT4fwpMeJf36POk6yJV_adQssw5c",

"consents_token":"eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiaWF0IjoxNTE2MjM5MDIyfQ.SflKxwRJSMeKKF2QT4fwpMeJf36POk6yJV_adQssw5c",

"request_id": "040dbfd1-40db-4caa-96ea-bf3c7d230877"

}

## 5.7 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 set format alongside it's description and register this directly to the consent & authorisation service via the UMA protection API. If no match is found then there is no further action required by the data provider.

## 5.8 Error handling

If the find request sent from the PFS fails then 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 which 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 must 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:<br><br>• PFS is not sending a HTTP POST<br>• parsing error by data provider<br>• 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 must 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 temporary used when the service is down for maintenance or is overloaded by requests |

# 6. Obtain PAT API

## 6.1 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 a OAuth2 authorisation grant, expressed as a JWT (JSON web token) which 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.

The obtain PAT API (i.e. the authorisation server's OAuth2 token endpoint) will be exposed by the authorisation server and will follow a standard implementation of https://datatracker.ietf.org/doc/html/rfc6749#section-4.5 using urn:ietf:params:oauth:grant-type:jwt-bearer in accord with https://datatracker.ietf.org/doc/html/rfc7523#section-2.1

## 6.2 Hosting

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

## 6.3 Format

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

## 6.4 Authorisation

The data provider will be a client of the authorisation server and will need to register it's 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.

## 6.5 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

## 6.6 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.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiaWF0IjoxNTE2MjM5MDIyfQ.SflKxwRJSMeKKF2QT4fwpMeJf36POk6yJV_adQssw5c”,

“token_type”:“pension_dashboard_pat”

}

## 6.7 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 authorized 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 recognize, 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",

}
```

# 7. Register PeI API

## 8.1 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.

## 8.2 Hosting

The API will be hosted on the authorisation server which is part of the 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.

## 8.3 Format

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

## 8.4 Authorisation

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

## 8.5 HTTP Method

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

## 8.6 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 URN of the resource, i.e. 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. It is a URN of the form: 'urn:pei:'<holder-name GUID>':'<asset GUID>'

**Type** – required

This is a URI of the type of all 'name' parameters used in the Pension Dashboard 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 dashboard 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 TBC.

## 8.7 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.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiaWF0IjoxNTE2MjM5MDIyfQ.SflKxwRJSMeKKF2QT4fwpMeJf36POk6yJV_adQssw5c

...

{

"resource_scopes":["value", “owner”, “delegate”],

"name": "pei:0e55140a-87d3-41cf-b6f7-bc822a4c3c3b:6e29eeb8-814c-44a6-a43f-b4830f3f4590",

"type": “http://pdp.gov/uma/PEI”,

“description” : “Aviva Pension”

}

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 an resource_id parameter. The resource _id parameter is issued by the authorisation server for each registered resource, i.e. 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 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"

}

## 8.8 Read resource description

The resource server must use 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.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiaWF0IjoxNTE2MjM5MDIyfQ.SflKxwRJSMeKKF2QT4fwpMeJf36POk6yJV_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:0e55140a-87d3-41cf-b6f7-bc822a4c3c3b:6e29eeb8-814c-44a6-a43f-b4830f3f4590",

"type": “http://pdp.gov/uma/PEI”,

“description” : “Aviva Pension”

}

## 8.9 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 an resource_id parameter.

Example of a 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.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9
lIiwiaWF0IjoxNTE2MjM5MDIyfQ.SflKxwRJSMeKKF2QT4fwpMeJf36POk6yJV_adQssw5c

...

{

  "resource_scopes":["value", “owner”, “delegate”],

  "name": "pei:0e55140a-87d3-41cf-b6f7-bc822a4c3c3b:6e29eeb8-814c-44a6-a43f-
b4830f3f4590",

  "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"

}
```

## 8.10 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.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiaWF0IjoxNTE2MjM5MDIyfQ.SflKxwRJSMeKKF2QT4fwpMeJf36POk6yJV_adQssw5c

...
```

Form of a successful response:

```
HTTP/1.1 204 No content

...
```

## 8.11 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 synchronization with the authorization server's understanding.

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

```
GET /rreg/ HTTP/1.1

Authorisation: Bearer
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiaWF0IjoxNTE2MjM5MDIyfQ.SflKxwRJSMeKKF2QT4fwpMeJf36POk6yJV_adQssw5c

...
```

Form of a successful response:

```
HTTP/1.1 200 OK

...
```

```
[ "KX3A-39WE",
  "9UQU-DUWW"
]
```

## 8.12 Error handling

If the request fails because the resource server does not have a valid access token (ie 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.

# 9. View API

## 9.1 Summary of view API

Enables a dashboard (client) to retrieve pension details on behalf of a requesting party (i.e 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.

## 9.2 Hosting

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

## 9.3 Format

The view 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 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:

- 'urn:pei:'<holder-name GUID>':'<asset GUID>'
- ?user=owner (the default if absent) or ?user=delegate (for advisers or guidance staff)

The asset GUID 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 create resource operation. The data provider will use the PAT associated with the pension owner to coordinate with the authorisation server to introspect the RPT (Section 6) or to obtain a permissions ticket (Section 7) if necessary.

Example of a view request carrying an RPT:

GET/pei/b1c832df301a431aab330c7ef88275de?user=owner HTTP/1.1

Host: www.dashboard.aviva.com

Authorisation: Bearer
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiaWF0IjoxNTE2MjM5MDIyfQ.cThIIoDvwdueQB468K5xDc5633seEFoqwxjF_xSJyQQ.

In the above example the asset GUID is b1c832df-301a-431a-ab33-0c7ef88275de.

## 9.6 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.eyJwZW5zaW9uQXJyYW5nZW1lbnREZXRhaWxzIjp7InBlbnNpb25SZWZlcmVuY2UiOiIwMDAzNzQ2MiIsInBlbnNpb25OYW1lIjoiU09MQVIgRU5FUkdZIFNZU1RFTVMgUEVOU0lPTiBGVU5EIiwicGVuc2lvblR5cGUiOiJEQiIsInBlbnNpb25PcmlnaW4iOiJXIiwicGVuc2lvblN0YXR1cyI6IkEiLCJwZW5zaW9uU3RhcnREYXRlIjoiMjAwNC0xMC0yMyIsInBlbnNpb25SZXRpcmVtZW50RGF0ZSI6IjIwNDUtMDctMDYifX0.G7oYVo1d18N-Y1TVBN-db9oruAjQNHureeM62hsPvOc"

}

## 9.7 Error Handling

If the view request is invalid the data provider will respond back to the dashboard with a 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 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.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiaWF0IjoxNTE2MjM5MDIyfQ.cThIIoDvwdueQB468K5xDc5633seEFoqwxjF_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.

# 10. Introspect API

## 10.1 Summary of Introspect API

When a resource server receives a view request from a dashboard which is a 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 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.

## 10.2 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.

## 10.3 Format

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

## 10.4 Authorisation

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

## 10.5 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.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9
lIiwiaWF0IjoxNTE2MjM5MDIyLCJhdWQiOiIxMjIzNDg5MCIsImV4cCI6MTIzNDU2Nzh9.XliAWL97B
MJx4V3WlIZESvEWhw7DGGUTJAOpIGYv_H0

...
```

token=eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiaWF0IjoxNTE2MjM5MDIyfQ.cThIIoDvwdueQB468K5xDc5633seEFoqwxjF_xSJyQQ

## 10.6 Response

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

**active**

REQUIRED. Boolean indicator of whether or not the presented token is currently active.

**permissions**

REQUIRED. Extension parameter named permissions that contains an array of objects, each one (representing a single permission) containing these parameters:

**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. Pension Dashboard valid granted scopes (in a Requesting Party Token, RPT) must be a list of two containing “value” and “owner” or ”delegate”.

**exp**

OPTIONAL. Integer timestamp, measured in the number of seconds since January 1 1970 UTC, indicating when this permission will expire. If the token-level exp value pre-dates a permission-level exp value, the token-level value takes precedence.

**token_type**

OPTIONAL. Type of the token as defined by PDPs UMA Profile, will be pension_dashboard_rpt

**exp**

OPTIONAL. Integer timestamp, measured in the number of seconds since January 1 1970 UTC, indicating when this token will expire. If the token-level exp value pre-dates a permission-level exp value, the token-level value takes precedence.

**iss**

OPTIONAL. String representing the issuer of this token.

Example of a response containing the introspection object:

HTTP/1.1 200 OK

Content-Type: application/json;charset=UTF-8

...


```
{
   "active": true,
   "permissions": [
      {
         "resource_id": "658b9e38-dd91-4e35-93ca-5154aba7321e0",
         "resource_scopes": [
            "value", “owner”
         ],
         "exp": 1651001341
      }
   ],
   "token_type": " pension_dashboard_rpt",
   "exp": 1651001341,
   "iss":https://claimsgathe.sandbox.k8s.dev.pensiondashboard.org/am/oauth2
   }
```


Resource Servers are responsible for access to the resource – the introspection response needs to be compared with what is stored internally (ie resource_id, scopes and expiry times) in order to determine whether the access attempt is valid and relates to the correct resource.

### 10.7 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 authorization server MUST respond with the HTTP 400 (Bad Request) status code and MAY respond with an invalid_request error code.

## 11. Permission API

### 11.1 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 it's view request.

### 11.2 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.

### 11.3 Format

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

### 11.4 Authorisation

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

### 11.5 HTTP method

The resource server will call the permission API using HTTP POST method. The body of the HTTP request message contains a JSON object for requesting a permission for 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. Pension Dashboard valid requested scopes (in a permission ticket) must be a list of two containing “value” and “owner” or ”delegate”.

Example of an HTTP request for a single permission at the authorization 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.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiaWF0IjoxNTE2MjM5MDIyLCJhdWQiOiIxMjIzNDg5MCIsImV4cCI6MTIzNDU2Nzh9.XliAWL97BMJx4V3WIIZESvEWhw7DGGUTJAOpIGYv_H0

...

{

  "resource_id":" KX3A-39WE",

  "resource_scopes":["value”, “owner”],

}
```

## 11.6 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 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.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiaWF0IjoxNTE2MjM5MDIyLCJhdWQiOiIxMjIzNDg5MCIsImV4cCI6MTIzNDU2Nzh9.XliAWL97BMJx4V3WlIZESvEWhw7DGGUTJAOpIGYv_H”

}

The PMT token is a JWT[3] as defined in [JWT] and contains the following claims.

**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 Resource Owner identifier at the AS *(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.

[3] 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

**jti**

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

**Public claim name**. *none*

**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 RO at the Resource Server. *(Derived from the PAT used in the initial permission ticket request.) May be of use to the RS 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.

**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 AS. Claim must be present when the permission ticket is presented for the second or subsequent time by a dashboard client. The AS 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 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 AS.

The token may be persisted by the AS 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 DB 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.

### 11.7 Error handling

If the resource server's permission registration request is authenticated properly but fails due to other reasons, the authorization 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 authorization 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.

## 12. PAT refresh API

### 12.1 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 and 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.

### 12.2 Hosting

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

### 12.3 Format

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

### 12.4 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 are no additional API security requirements for the refresh API.

### 12.5 HTTP method

The authorisation server will be restricted to only make a HTTP POST requests 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 which 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 C&A 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.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiaWF0IjoxNTE2MjM5MDIyfQ.SflKxwRJSMeKKF2QT4fwpMeJf36POk6yJV_adQssw5c”,

"consents_token":"eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiaWF0IjoxNTE2MjM5MDIyfQ.SflKxwRJSMeKKF2QT4fwpMeJf36POk6yJV_adQssw5c",

"resource_id": “KX3A-39WE”,

“pei”:“pei:aed123aq:WGF45920EJH348ASEWQ0284"

}

## 12.6 Response

If the request sent by the C&A is success then the data provider will respond with a 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 the authorisation server token endpoint – see Section 3 Obtain PAT.

## 12.7 Error handling

If the PAT refresh request sent by the authorisation server is fails then 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 verbose and plain language where it describes, accordingly with the error catching process, what went wrong and possibly how to amend 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:<br><br>• authorisation server is not sending a HTTP POST<br>• Parsing error by data provider<br>• 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 | |

# 13. Authorise API

## 13.1 Summary of authorise API

The dashboard initiates the authorisation 'dance' using this Oauth specific API to 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 it's view requests in the future.

## 13.2 Hosting

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

## 13.3 Format

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

## 13.4 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.

## 13.5 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 (i.e. for the combination of the client and 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 RS 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.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiaWF0IjoxNTE2MjM5MDIyfQ.SflKxwRJSMeKKF2QT4fwpMeJf36POk6yJV_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.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiaWF0IjoxNTE2MjM5MDIyLCJleHAiOjM1Nzg5Nzg5NywianRpIjo1NDIzNjIzOTgwfQ.hp1udPrSdTRkLf3QzRoHFff_T6BzRqlSADIZoZ-c5sI

## 13.6 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.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiaWF0IjoxNTE2MjM5MDIyfQ.SflKxwRJSMeKKF2QT4fwpMeJf36POk6yJV_adQssw5c ",

“token_type”: “pension_dashboard_rpt”,

“upgraded”: false,
"pct":"eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiaWF0IjoxNTE2MjM5MDIyLCJleHAiOjM1Nzg5Nzg5NywianRpIjo1NDIzNjIzOTgwfQ.hp1udPrSdTRkLf3QzRoHFff_T6BzRqlSADIZoZ-c5sI"

}

## 13.7 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.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiaWF0IjoxNTE2MjM5MDIyLCJleHAiOjM1Nzg5Nzg5NywianRpIjo1NDIzNjIzOTgwfQ.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"

}

# 14. Obtain PeIs API

## 14.1 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 C&A account via an API hosted at the authorisation server.

## 14.2 Hosting

The API (an UMA Resource Server) will be hosted on the authorisation server which is part of the C&A service.

## 14.3 Format

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

## 14.4 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.

## 14.5 HTTP Method

The Dashboard will make a HTTP GET request to the obtain PeI endpoint for that specific user's C&A 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 C&A 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/a7f542a22c8647b3b62c2fb9a81c2495?user=owner HTTP/1.1

Host: www.CA.ObtainPeIs.com

Authorization: Bearer
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiaWF0IjoxNTE2MjM5MDIyfQ.cThIIoDvwdueQB468K5xDc5633seEFoqwxjF_xSJyQQ.

## 14.6 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 C&A 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: 9d0f04b2-8a47-4fb9-91d9-08828036b631:463aabf7-75c1-4d87-b5c8-2c7ffca2341f", "pei: f315e508-0c84-41bf-afdf-e40db1cb10ec:9da49adb-bc27-4904-8303-ba367ead8592", "pei: 728f9722-88c1-42f3-965a-d2faab8967e8:26d93ebc-0dfd-43c0-bfee-2b8f8ad7a742"]

}

## 14.7 Error handling

If the obtain PeI request is invalid the C&A 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 C&A 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.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiaWF0IjoxNTE2MjM5MDIyfQ.cThIIoDvwdueQB468K5xDc5633seEFoqwxjF_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.

# 15. Obtain PeI configuration API

## 15.1 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':<holder-name GUID>':'<asset GUID> e.g."pei:cd2a3015-090e-469e-839b-5e5435a29512:3d2b0cde-5831-4537-b4e4-d6c44bf1373a") and breaking it into components, looking up the holdername, e.g. 'cd2a3015-090e-469e-839b-5e5435a29512' in a configuration table to derive a scheme name, eg 'aviva.dashboard/pei', and composing a URL, e.g.https://aviva.dashboard/pei/3d2b0cde58314537b4e4d6c44bf1373a. 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.

## 15.2 Hosting

The API will be hosted on the governance register

## 15.3 Format

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

## 15.4 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.

## 15.5 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 (i.e. holdername) – ie the dashboard attempts to GET the URL https:://GR.ObtainPeIsConfig?pei"= 26d93ebc-0dfd-43c0-bfee-2b8f8ad7a742, 728f9722-88c1-42f3-965a-d2faab8967e8,..." 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.

## 15.6 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

...

 {

"26d93ebc-0dfd-43c0-bfee-2b8f8ad7a742": "aviva.dashboard/pei",

"728f9722-88c1-42f3-965a-d2faab8967e8": "scottishwidows.dashboard/pei"

}
```

## 15.7 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

# 16. Technical standards

## 16.1 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 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 (incl pullPeis)
- refresh PAT (RS failure case)
- consent
- account deletion (GDPR)

For every interaction made by the dashboard to the C&A it has to mint a new RQP token.

Formal description of the token. The RqP token is a JWT as defined in [JWT] and must profiled as follows.

REQUIRED iss. Registered claim name. Defined [JWT]. Profiled: unique identifier within dashboard ecosystem of the dashboard instance issuing the JWT. (This is Dbi.)

REQUIRED sub. Registered claim name. Defined [JWT]. Profiled: unique identifier within scope of iss, of the requesting party which is authenticated to iss at the time the JWT is issued. (This is user@dbi.)

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

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

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

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

Public claim name. none

REQUIRED role. Private claim name. The iss states the role in which the requesting party is acting. String value. One of "owner" or "delegate".

The token MUST be signed by the issuer. The token MAY be encrypted for the AS.

The token must always be presented to the token or claims interaction endpoints at the AS by the DB client along with an AS issued token. It does not need to be bound.

## 16.2 JWT signing and verification

After successfully onboarding to the CDA you will receive a crypto package which will contain a signing certificate and embedded within it the private key which will be used to generate the signature using RS256 as the signing algorithm to sign a JWT.

In order verify the signature of a JWT the CDA will expose a centralised JWKS endpoint which you will able to call to obtain the corresponding public key to verify the signature.

## 16.3 Pension identifier format

The pension identifier (PeI) shall have a standardised format across all providers. It is expressed in the form of a URN (uniform resource name) that provides a location-independent, globally-unique, persistent identifier, with a defined namespace. PeIs are issued by the data provider and must be associated with a matched (full or possible) pension asset. PeIs (more specifically the PeI.assetGUID) can be associated with a pension asset at any time prior to registration of the asset with the C&A Service. Once associated with a pension asset a PeI must not be reused for a different pension asset.

It is a URN  of the form: 'urn:pei:'<holder-name GUID>':'<asset GUID>.

An example of a possible PeI is:

urn:pei:f1c72611-438b-4f72-a4b5-ec7e69000c31:8ff2063a-48bd-4ed7-bcf8-7c3b8f89626d

Formal description of the PeI. It must be profiled as follows.

REQUIRED

Both the Holder-name GUID and the Asset GUID are globally unique identifiers

Holder-name is a globally unique string which can be dereferenced to an endpoint (i.e. URL) which serves view requests, composed of the view endpoint and the asset GUID making the query URL. If properly authorised, the pension details associated with that asset ID are served by the full URL.

urn:pei:f1c72611-438b-4f72-a4b5-ec7e69000c31:8ff2063a-48bd-4ed7-bcf8-7c3b8f89626d

example View URL constructed by dereferencing the holdername to the base URL :

https://testISP.co.uk/8ff2063a48bd4ed7bcf87c3b8f89626d

## 16.4 GUID creation protocols

Formal description of GUIDs. They are 32 hex digits (128 bits) allocated 'randomly' by standard methods profiled using the approach in rfc4122 (https://www.ietf.org/rfc/rfc4122.txt).

## 16.5 Data providers (UMA Resource Servers)

Data providers (UMA resource servers) will operate the UMA FedAuthz protocols, which include introspection, permission and register API calls, as described in the above sections.

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 the Protection 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 allows it to locate them using the inbound URL of the view request.

Resource servers are responsible for access to the resource – introspection response needs to be compared with what is stored internally for the resource in order to ensure access request is authorised.

# 17. Appendix

## 17.1 Glossary

| Term | Definition |
|---|---|
| Resource Owner<br><br>(i.e. Pension Owner) | The resource owner is a user or legal entity that is capable of granting access to a protected resource |
| Client (i.e. 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 C&A – 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 -<br><br>Permission Token | Gives permission for a dashboard to initiate the authorisation protocol in order to authorise a retrieval request |
| RQP –<br><br>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 –<br><br>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 (i.e. one to one association) |

| | |
|---|---|
| PCT –<br><br>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 –<br><br>Protection API Token | OAuth2 token scope UMA protection which represents the Resource Owner's authorisation for the RS to manage federated authorisation at the AS (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 |