

# Plan Management (Dark Matter)

## Overview

Plan management deals with the organization and management of users for B2B customers.

## What We Own

- ☐ <https://github.com/ps-dev/plan-management> - Connect your Github account
- ☐ <https://github.com/ps-dev/ps-mui> - Connect your Github account
- ☐ <https://github.com/ps-dev/plans-account-switcher> - Connect your Github account
- ☐ <https://github.com/ps-dev/plans-org-structure-selection> - Connect your Github account

## Docs

- [Plan Management \(Dark Matter\) Definitions](#)
- [Plan Management \(Dark Matter\) Public Engineering Docs](#)
- [Documentation for the Plans-Org-Structure-Selection Package](#)
- [Plan Management \(Dark Matter\) Product](#)
- [Plan Management \(Dark Matter\) Private Dev Team Docs](#)

# Active Definitions

## ***Plan***

Identified by a `planId`.

A collection of users, product licenses, teams, and leaders. This often maps to a company or organization.

## ***User***

Identified by a `userHandle`.

Any person who uses Pluralsight.

## ***Plan User***

Identified by a `planUserId` or a combination of a `planId` and `userHandle`.

A user who is associated with a plan. Plan users can be either redeemed or pending. A redeemed plan user is a user with a Pluralsight account. A pending plan user has had an invite sent, but has not yet redeemed the invitation.

## ***Member (alias: Learner)***

Identified by a `memberId`, `learnerId`, or a combination of a `planId` and `userHandle`.

A plan user with access to content via a license. The term "member" is being deprecated in favor of "learner." "Member" has previously been used to describe both a plan user and a plan user with a content license. To fix this ambiguity, we are using "plan user" for users who are on the plan and "learner" for plan users who consume one or

more licenses going forward. For now, you may occasionally see the term "member" used for both of these groups.

## ***Invite (alias: Invitation)***

Identified by a `planUserId` when using the [DVS](#). If using the old [RabbitMQ](#) messages, they are identified by an `inviteId`.

A notification sent to a user informing them they have been granted access to Pluralsight with a link to redeem access. A pending plan user gets created for each user (email) an invite has been sent to. Upon redeeming the invitation, the plan user is converted from pending to redeemed, and the user account is associated with the plan user.

## ***License***

Grants access to a product (ex. Skills, Cloud Labs). Plan users can be given a license to become a learner.

## ***Team***

Identified by a combination of a `planId` and `teamId`.

A named collection of plan users. Teams may be nested i.e. a team may contain one or more teams.

## ***Leader***

Identified by a `teamManagerId` or a combination of a `planId`, `teamId`, and a `userHandle`.

A plan user who manages other plan users. Their abilities are defined by one or more permission sets. Leaders have access to analytics and can manage content for their users.

## ***Permission set***

Identified by a `permissionSetId`.

A collection of abilities and controls given to a leader. A plan contains multiple permission sets. Leaders may have multiple permission sets.

## ***Manager***

Identified by a `teamManagerId` or a combination of a `planId`, `teamId`, and a `userHandle`.

A leader who manages one or more teams. Managers only have purview over plan users on teams they manage and unassigned plan users (users not assigned to any team).

## ***Admin***

Identified by a `teamManagerId` or a combination of a `planId`, `teamId`, and a `userHandle`.

A leader who can perform any action on a plan. In addition to their other abilities, only admins can access billing and create priorities for their plan.

# Deprecated Definitions

\*Included if you find yourself referencing out of date documentation or using older data sources.

## ***Department***

A top-level container for organizing users. Deprecated in favor of nested teams.

## ***Group***

A container for organizing users, nested inside of a department. Deprecated in favor of nested teams.

[Plan Management Eventing Organized by Entity](#)

[Plan Management Internal API Endpoints](#)

[Plan Management RabbitMQ Queues](#)

[Plan Management DVS Topics](#)

## **Plan**

Identified by a `planId`.

- DVS topics:
  - [exp.plans.Plan](#)
  - [exp.plans.v1.PlanMerge](#)
- RabbitMQ queues:
  - [ps.plans.plan-updated.v5](#)
  - [ps.plans.plan-archived.v1](#)
- Internal API endpoints:
  - [Plans API: Create new plan](#)
  - [Plans API: Get all plans](#)
  - [Plans API: Get plan details](#)

## **Plan User**

Identified by a `planUserId` or a combination of a `planId` and `userHandle`.

- DVS topics:
  - [exp.plans.v1.PlanUser](#)
  - [exp.plans.v1.PlanUserPermissionSet](#)
  - [exp.plans.v1.PlanUserTeam](#)
- RabbitMQ queues:
  - [ps.plans.user-authorization-updated.v3](#)
- Internal API endpoints:
  - [Plans API: Get user details](#)
  - [Plans API: Get user authorization details](#)
  - [Plans API: Remove member from expired plan](#)
  - [Plans API: Update member note](#)

## Team

Identified by a combination of a `planId` and `teamId`.

- DVS topics:
  - [exp.plans.v3.Team](#)
  - [exp.plans.v1.PlanUserTeam](#)
- RabbitMQ queues:
  - [ps.plans.team-updated.v3](#)
  - [ps.plans.team-removed.v2](#)
  - [ps.plans.member-assigned-to-team.v1](#)
  - [ps.plans.member-removed-from-team.v1](#)
- Internal API endpoints:
  - [Plans API: Create team](#)
  - [Plans API: Rename team](#)

- [Plans API: Change team description](#)
- [Plans API: Change team parent](#)
- [Plans API: Dissolve team](#)
- [Plans API: Add member to team](#)
- [Plans API: Move member to team](#)
- [Plans API: Remove member from team](#)

## **Member / Learner**

Identified by a `memberId`, `learnerId`, or a combination of a `planId` and `userHandle`.

- DVS topics:
  - [exp.plans.v3.Member](#)
  - [exp.plans.v5.Membership](#)
- RabbitMQ queues:
  - [ps.plans.member-added-to-plan.v1](#)
  - [ps.plans.member-removed-from-plan.v1](#)
  - [ps.plans.member-assigned-to-team.v1](#)
  - [ps.plans.member-removed-from-team.v1](#)
  - [ps.plans.member-note-updated.v1](#)
- Internal API endpoints:
  - [Plans API: Get all members for a plan](#)
  - [Plans API: Get member details](#)
  - [Plans API: Invite member](#)
  - [Plans API: Update member note](#)
  - [Plans API: Update member products](#)
  - [Plans API: Remove member](#)
  - [Plans API: Add member to team](#)

- Plans API: Move member to team
- Plans API: Remove member from team

## Team Manager

Identified by a `teamManagerId` or a combination of a `planId`, `teamId`, and a `userHandle`.

- DVS topics:
  - `exp.plans.Manager`
- RabbitMQ queues:
  - `ps.plans.team-manager-added.v4`
  - `ps.plans.team-manager-removed.v3`
  - `ps.plans.user-authorization-updated.v3`
- Internal API endpoints:
  - Plans API: Add team manager (from existing user)
  - Plans API: Remove team manager
  - Plans API: Invite team manager

## Permission Set

Identified by a `permissionSetId`.

- DVS topics:
  - `exp.plans.v1.PermissionSet`
  - `exp.plans.v1.PlanUserPermissionSet`

## Invite

Identified by a `planUserId` when using the DVS. If using the old RabbitMQ messages, they are identified by an `inviteId`.

- DVS topics:



- [exp.plans.v2.MemberInvite](#)
- RabbitMQ queues:
  - [ps.plans.invite-created.v4](#)
  - [ps.plans.invite-redeemed.v4](#)
  - [ps.plans.invite-canceled.v3](#)
- Internal API endpoints:
  - Plans API: Invite team manager
  - Plans API: Cancel invite
  - Plans API: Invite member
  - Plans API: Update invite note

## Plan Management Internal API Endpoints



Owned by [Gemma Grover](#)

Last updated: [Feb 24, 2023](#) 2 min read 34 people viewed

### **Plan API Endpoints**

### **User/Plan User API Endpoints**

### **Member (alias: Learner) API Endpoints**

### **Team API Endpoints**

### **Invite Endpoints**

## Definitions to Keep In Mind

### **Plan**

**Identified by a planId.**

*A collection of users, product licenses, teams, and leaders. This often maps to a company or organization.* [User/Plan User](#)

## User vs Plan User

**User is identified by a userHandle.**

**Plan User is identified by a planUserId or a combination of a planId and userHandle.**

*A user is any person who uses Pluralsight. A plan user is a user who is associated with a plan. Plan users can be either redeemed or pending. A redeemed plan user is a user with a Pluralsight account. A pending plan user has had an invite sent, but has not yet redeemed the invitation.*

## Learner (Formerly Known as *Member*)

**Identified by a memberId, learnerId, or a combination of a planId and userHandle.**

*A plan user with access to content via a license. The term "member" is being deprecated in favor of "learner." "Member" has previously been used to describe both a plan user and a plan user with a content license. To fix this ambiguity, we are using "plan user" for users who are on the plan and "learner" for plan users who consume one or more licenses going forward. For now, you may occasionally see the term "member" used for both of these groups.*

## Leader (Team Managers and Admins)

**Identified by a teamManagerId or a combination of a planId, teamId, and a userHandle.**

*A plan user who manages other plan users. Their abilities are defined by one or more permission sets. Leaders have access to analytics and can manage content for their users.*

## Team

**Identified by a combination of a planId and teamId.**

*A named collection of plan users. Teams may be nested i.e. a team may contain one or more teams.*

## Plan API Endpoints

[Create New Plan](#)

[Get All Plans](#)

[Get Plan Details](#)

## Create Plans

### Request Details

**HTTP Method:** POST

**URI Template:**

- Staging: <https://plans-api-stage.vnerd.com/v5/plans>
- Production: <https://plans-api.vnerd.com/v5/plans>

**Headers:**

- Authorization: Token <value>


### Request Payload

```
{  
  "data": {  
    "entitlementsId": "2c92a0fc5965993cfc89f0567",  
    "cloudOrgId": "someCloudOrgId_ThisFieldIsOptional",  
    "displayName": "Test Company",
```

```
"adminUsers": [{
  "email": "testAdmin@example.com",
  "firstName": "John",
  "lastName": "Doe",
  "handle": "1d42d41d5d414d24d41d245"
}],
"products": [
  {
    "productId": "b6d2580a-6ba2-4569-8734-f7428c4d370c",
    "productOptionId": "a3ed696d-fc8f-4d2b-a8d3-4d09a49885fc",
    "startsAt": "2022-04-21T07:00:00.000Z",
    "expiresAt": "2023-04-21T07:00:00.000Z",
    "inTerm": true,
    "quantity": 1000
  },
  {
    "productId": "036ebb99-f23a-470d-835d-a66272862556",
    "productOptionId": "277fdc0e-23f6-4ca0-9bde-bdf2350a91ca",
    "startsAt": "2022-04-21T07:00:00.000Z",
    "inTerm": true,
    "quantity": 10
  }
]
```

```
}
```

## Token authorization

You'll need to get a token to use this API. Talk to @dark-matter-dev in   
<https://pluralsight.slack.com/archives/C8ZCWDZ5H> - Connect your Slack account to get one.

Put your token in an authorization header: `Authorization: Token <value>`

## Response

Successful (200) Response:

```
{  
  "data": {  
    "planId": "new-plan-id-84fd5"  
  }  
}
```

Plan Already Exists (400) Response:

```
{  
  "errors": [  
    {  
      "detail": "This plan already exists.",  
      "status": "400",  
      "code": "plan-already-exists"  
    }  
]
```

```
]
}
```

Unprocessable Entity (422) Response:

```
{
  "errors": [
    {
      "detail": "Products should not be empty.",
      "status": "422",
      "code": "missing-required-fields"
    }
  ]
}
```

## Changes

- Changed field `zuoraAccountId` to `entitlementsId`
- Added `cloudOrgId` field

## Get All Plans

### Request Details

Returns a paginated list of all plans, including expired plans.


The page size is 50. Use the next link to retrieve the next page. When there are no more pages, next will be null.

**HTTP Method:** GET

**URI Template:**

- Staging: <https://plans-api-stage.vnerd.com/v4/plans>
- Production: <https://plans-api.vnerd.com/v4/plans>

**Headers:**

- Authorization: Token <value>
- You'll need to get a token to use this API. Talk to @dark-matter-dev in  <https://pluralsight.slack.com/archives/C8ZCWDZ5H> - Connect your Slack account to get one.

## Response

```
{
  "data": [
    {
      "planId": "morningstar",
      "subscriberId": null,
      "displayName": "Morningstar, Inc.",
      "createdOnUtc": "2015-01-01T00:00:00+00:00",
      "expiresOnUtc": "2017-01-01T00:00:00+00:00",
      "totalLicenseCount": 10,
      "isPilot": false
    }
  ]
}
```

```
},
{
  "planId": "plural-a8e21",
  "subscriberId": "101079",
  "displayName": "plural",
  "createdOnUtc": "2016-10-18T16:38:27.196+00:00",
  "expiresOnUtc": "2017-10-18T00:00:00+00:00",
  "totalLicenseCount": 20,
  "isPilot": false
},
{
  "links": {
    "next":
      "https://plans-api-stage.pluralsight.com/v1/plans?next=-9099229994786156941"
  }
}
```

## Changes

- None

## Get Plan Details



# Request Details

Given a `planId`, returns the plan details or a 404 if not found.


This includes plan metadata, teams, and team managers. To get all members for a plan, use the [Get all learners for a plan](#) endpoint.

**HTTP Method:** GET

**URI Template:**

- Staging: <https://plans-api-stage.vnerd.com/v4/plans/:planId>
- Production: <https://plans-api.vnerd.com/v4/plans/:planId>

**Headers:**

- Authorization: Token <value>
- You'll need to get a token to use this API. Talk to @dark-matter-dev in   
<https://pluralsight.slack.com/archives/C8ZCWDZ5H> - Connect your Slack account to get one.

# Response

```
{
```

```
"data": {
```

```
"planId": "1800contacts",
```

```
"displayName": "Microsoft C+E Specialist Sales",
```

```
"createdOnUtc": "2015-01-01T00:00:00+00:00",
```

```
"expiresOnUtc": "2020-07-07T00:00:00+00:00",

"isPilot": false,

"totalLicenseCount": 100,

"products": [

{

  "id": "4a4b5b26-70b1-4910-b854-79c67259a09a",

  "name": "Business Enterprise",

  "purchasedLicenseCount": 1000,

  "assignedLicenseCount": 750,

  "remainingLicenseCount": 25

},

{

  "id": "7e659c2e-a4df-448d-b2a4-17012cc76fe7",

  "name": "ITIL 4 Foundation",

  "purchasedLicenseCount": 200,

  "assignedLicenseCount": 50,

  "remainingLicenseCount": 150

}
```

```
],  
  
"isSuspended": false,  
  
"activeLearnerInviteCount": 27,  
  
"teams": [  
  
  {  
  
    "id": "73eedaf-00e1-4550-9450-fa3b23f598fe",  
  
    "parentTeamId": "85ueedaf-00e1-4550-8756-fa3b23f598io",  
  
    "name": "Software Engineering",  
  
    "managers": [  
  
      {  
  
        "id": "44214b9b-fde1-48a0-ba31-1051cb146258",  
  
        "userHandle": "11214b9b-41e1-48a0-ba31-1051cb146925"  
  
      }  
  
    ]  
  
  },  
  
  {  
  
    "id": "bb8c695c-2139-461a-8881-8d6316bfdccd",  
  
    "parentTeamId": null,
```

```
"name": "Turing",

"managers": []

},

],

"features": {

"advancedChannelAnalytics": false,

"advancedSkillAnalytics": false,

"advancedRoleIqAnalytics": false,

"nonTechAnalytics": true,

"roleIqAnalytics": true,

"channelAnalytics": true,

"analyticsUpsell": true,

"qAndADownload": false,

"transcender": true,

"priorities": false,

"projects": false,

"customRoleIq": false,

"skillsStrategy": false,
```

```
"labs": false,  
  
"itil": false,  
  
}  
  
}  
  
}
```

## Changes

### Breaking changes from **v3**:

- Collapsed features and restrictions fields into just features and renamed some features to be more consistent
  - nonTechAnalyticsRestricted renamed to nonTechAnalytics and value inverted
  - analyticsUpsellRestricted renamed to analyticsUpsell and value inverted
  - advancedSkillsReporting renamed to advancedSkillAnalytics
  - channelsReportingRestricted renamed to channelAnalytics and value inverted
  - advancedChannelsReporting renamed to advancedChannelAnalytics
  - roleIQAnalyticsRestricted renamed to roleIqAnalytics and value inverted
  - roleIQAnalytics renamed to advancedRoleIqAnalytics
  - gitPrimeAnalytics removed
- Removed features and moved them to user-level authorization data

- userSkillLevelRestricted
  - licenseAllocationSummaryRestricted
- Collapsed teams, groups, and departments into just teams
  - Renamed teamId field to id
  - Removed groupId field and replaced with parentTeamId
- Removed admins field

## User/Plan User API Endpoints

[Get User/Plan User Details](#)

[Get User/Plan User Authorization Details](#)

[Track Last Plan Accessed for Leader](#)

[Remove user from Plan](#)

## Get User/Plan User Details

### Request Details

Given a `userHandle`, returns the manager and learner information for that user, or a 404 if the user is not a user on any plan. It is possible for a user to be on multiple plans and also a manager of multiple teams. A user can only be a learner(member) of a single plan.


This can be used to self-heal the messages [ps.plans.team-manager-added.v4](#), [ps.plans.member-added-to-plan.v1](#), and [ps.plans.member-assigned-to-team.v1](#).

**HTTP Method:** GET

**URI Template:**

- Staging: <https://plans-api-stage.vnerd.com/v4/users/:userHandle>
- Production: <https://plans-api.vnerd.com/v4/users/:userHandle>

**Headers:**

- Authorization: Token <value>
- You'll need to get a token to use this API. Talk to @dark-matter-dev in  <https://pluralsight.slack.com/archives/C8ZCWDZ5H> - Connect your Slack account to get one.

## Response

```
{
  "data": {
    "userHandle": "user-handle",
    "teamManagers": [
      {
        "id": "852b217f-ff4d-4770-a7e2-822fa57f5gt5",
        "planId": "adventureworks",
        "teamId": "913b217f-ff4d-4770-a7e2-822fa57f5bc8"
      },
      {
        "id": "9c625cc2-7c73-4822-9d5b-038bd6405af2",
        "planId": "pluralsight-engineering-1d36a",
        "teamId": "792277f4-dbff-41ee-9cfd-95c7b8cad9c1"
      }
    ],
    "member": {
      "id": "852277f4-dbff-41ee-9cfd-95c7b8cad8d1",
```

```
"planId": "pluralsight-engineering-1d36a",
"teamIds": [ "52f1611d-552c-4f57-a5ca-8d9b7e0c29f5" ],
"startDate": "2015-12-02T17:02:10.356+00:00",
"note": "some-note"
}
}
}
```

## Notes

- If the user is not a team manager or member, but is still on the plan, the `userHandle` will be populated with an empty array for `teamManagers` and null for `member`.
- If the user is not a team manager, the `teamManagers` array will be present but empty.
- If the user is not a member, the `member` property will be null.
- If the user is not on any plan a 404 will be returned.
- A user only consumes a license if they are a member.

## Changes

- removed `Admins`, as user authorization should be done from [Plans API: Get user authorization details](#)
- `teamId` on member is now `teamIds` with a list of all `teamIds` the member is on
- collapsed all types of `Managers` into `teamManagers`
- `userHandle` field instead of the user object

## Get User/Plan User Authorization Details

Given a `userHandle`, returns the SDL authorization information for that user.

This can be used to self-heal the message [ps.plans.user-authorization-updated.v3](#).

- Staging: <https://plans-api-stage.vnerd.com/v4/users/:userHandle/authz>
- Production: <https://plans-api.vnerd.com/v4/users/:userHandle/authz>



# Token authorization

You'll need to get a token to use this API. Talk to @dark-matter-dev in 

<https://pluralsight.slack.com/archives/C8ZCWDZ5H> - Connect your Slack account to get one.

Put your token in an authorization header: `Authorization: Token <value>`

## Summary

Users can be authorized at the plan or team level.

- If a user is authorized for a team, they're authorized for all teams nested under that team.
- If a user is authorized at the plan level, it is *assumed* they're authorized for all teams on that plan, and the response will *not* list them.
- Users can be authorized on more than one, and any combination of plan/team.
- Because of this, you cannot reverse engineer the nested team structure from this endpoint.

## Example Response for user who can view everything for a single plan

```
{
  "data": {
    "userHandle": "14812b48-3ba8-4396-ba0e-76375d262ed1",
    "plans": [
      {
        "id": "test-plan",
        "leader": {
          "managedTeams": [],
          "authz": {
```

```
"canViewDashboard": true,
"canViewAccount": true,
"canViewPeople": true,
"canViewAnalytics": true,
"canViewLogs": true,
"canViewSettings": false,
"canViewAllUsers": true,
"canViewBilling": true,
"canInviteUsers": true,
"canViewPriorities": true,
"canViewCustomRoleIq": true,
"canViewUserSkillLevels": true,
"canViewLicenseAllocations": true,
"canViewPrograms": true,
"canViewCustomSandboxes": true,
"canViewLearningPaths": true,
"canViewSkillsAssessments": true,
"canViewStudyGroups": true
}
}
}
],
"allPlansAuthz": {
"canViewDashboard": false,
"canViewAccount": false,
"canViewPeople": false,
"canViewAnalytics": false,
"canViewLogs": false,
"canViewSettings": false,
"canViewAllUsers": false,
"canViewBilling": false,
"canInviteUsers": false,
"canViewPriorities": false,
"canViewCustomRoleIq": false,
"canViewUserSkillLevels": false,
```

```
"canViewLicenseAllocations": false,
"canViewPrograms": false,
"canViewCustomSandboxes": false,
"canViewLearningPaths": false,
"canViewSkillsAssessments": false,
"canViewStudyGroups": false
},
"cacheUntil": "2019-02-16T18:20:57.8922367+00:00"
}
}
```

## Example response for user who can view a single team on a plan:

```
{
  "data": {
    "userHandle": "51646243-bb47-4681-be55-aabe48765879",
    "plans": [
      {
        "id": "test-plan",
        "leader": {
          "managedTeams": [
            {
              "teamId": "82d149e7-1cb2-4e76-ae4-345076a21101",
              "authz": {
                "canViewUsers": true
              }
            }
          ]
        },
        "authz": {
          "canViewDashboard": true,
          "canViewAccount": false,
          "canViewPeople": true,
          "canViewAnalytics": true,

```

```
"canViewLogs": false,
"canViewSettings": false,
"canViewAllUsers": false,
"canViewBilling": false,
"canInviteUsers": false,
"canViewPriorities": false,
"canViewCustomRoleIq": true,
"canViewUserSkillLevels": false,
"canViewLicenseAllocations": false,
"canViewPrograms": false,
"canViewCustomSandboxes": false,
"canViewLearningPaths": false,
"canViewSkillsAssessments": false,
"canViewStudyGroups": false
}
}
},
"allPlansAuthz": {
"canViewDashboard": false,
"canViewAccount": false,
"canViewPeople": false,
"canViewAnalytics": false,
"canViewLogs": false,
"canViewSettings": false,
"canViewAllUsers": false,
"canViewBilling": false,
"canInviteUsers": false,
"canViewPriorities": false,
"canViewCustomRoleIq": false,
"canViewUserSkillLevels": false,
"canViewLicenseAllocations": false,
"canViewPrograms": false,
"canViewCustomSandboxes": false,
"canViewLearningPaths": false,
```

```
"canViewSkillsAssessments": false,
"canViewStudyGroups": false
},
"cacheUntil": "2019-02-16T18:18:04.2130744+00:00"
}
}
```

Example response for user who can view everything for a team with two nested teams on a plan:

```
{
  "data": {
    "userHandle": "4c22a8ba-1357-47aa-a510-303565c65f87",
    "plans": [
      {
        "id": "test-plan",
        "leader": {
          "managedTeams": [
            {
              "teamId": "48d28fcb-8bed-43d7-b00f-c13af8fa4ad9",
              "authz": {
                "canViewUsers": true
              }
            },
            {
              "teamId": "0beba390-6da1-4b75-8d2d-0032d1b8b0f5",
              "authz": {
                "canViewUsers": true
              }
            },
            {
              "teamId": "82d149e7-1cb2-4e76-ae4-345076a21101",
```

```
"authz": {
  "canViewUsers": true
}
},
"authz": {
  "canViewDashboard": true,
  "canViewAccount": false,
  "canViewPeople": true,
  "canViewAnalytics": true,
  "canViewLogs": false,
  "canViewSettings": false,
  "canViewAllUsers": false,
  "canViewBilling": false,
  "canInviteUsers": true,
  "canViewPriorities": true,
  "canViewCustomRoleIq": true,
  "canViewUserSkillLevels": true,
  "canViewLicenseAllocations": true,
  "canViewPrograms": false,
  "canViewCustomSandboxes": false,
  "canViewLearningPaths": false,
  "canViewSkillsAssessments": false,
  "canViewStudyGroups": false
}
}
},
"allPlansAuthz": {
  "canViewDashboard": false,
  "canViewAccount": false,
  "canViewPeople": false,
  "canViewAnalytics": false,
  "canViewLogs": false,
  "canViewSettings": false,
```

```
"canViewAllUsers": false,
"canViewBilling": false,
"canInviteUsers": false,
"canViewPriorities": false,
"canViewCustomRoleIq": false,
"canViewUserSkillLevels": false,
"canViewLicenseAllocations": false,
"canViewPrograms": false,
"canViewCustomSandboxes": false,
"canViewLearningPaths": false,
"canViewSkillsAssessments": false,
"canViewStudyGroups": false
},
"cacheUntil": "2019-02-16T18:29:07.2141927+00:00"
}
}
```

## Example response for user who can view all plans:

```
{
  "data": {
    "userHandle": "58b405e0-08dd-4b0b-b94f-ea770f4cefff",
    "plans": [],
    "allPlansAuthz": {
      "canViewDashboard": true,
      "canViewAccount": true,
      "canViewPeople": true,
      "canViewAnalytics": true,
      "canViewLogs": true,
      "canViewSettings": true,
      "canViewAllUsers": true,
      "canInviteUsers": true,
      "canViewAllUsers": true,
    }
  }
}
```

```
"canViewBilling": false,
"canInviteUsers": true,
"canViewPriorities": true,
"canViewCustomRoleIq": true,
"canViewUserSkillLevels": true,
"canViewLicenseAllocations": true,
"canViewPrograms": true,
"canViewCustomSandboxes": false,
"canViewLearningPaths": false,
"canViewSkillsAssessments": false,
"canViewStudyGroups": false
},
"cacheUntil": "2019-02-16T18:39:34.7095756+00:00"
}
}
```

## Example response for user with no permissions:

```
{
  "data": {
    "userHandle": "58b405e0-08dd-4b0b-b94f-ea770f4cefff",
    "plans": [],
    "allPlansAuthz": {
      "canViewDashboard": false,
      "canViewAccount": false,
      "canViewPeople": false,
      "canViewAnalytics": false,
      "canViewLogs": false,
      "canViewSettings": false,
      "canViewAllUsers": false,
      "canInviteUsers": false,
      "canViewAllUsers": false,
      "canViewBilling": false,

```



```
"canInviteUsers": false,
"canConnectRepositories": false,
"canViewPriorities": false,
"canViewCustomRoleIq": false,
"canViewUserSkillLevels": false,
"canViewLicenseAllocations": false,
"canViewPrograms": false,
"canViewCustomSandboxes": false,
"canViewLearningPaths": false,
"canViewSkillsAssessments": false,
"canViewStudyGroups": false
},
"cacheUntil": "2019-02-16T18:41:03.0527411+00:00"
}
}
```

## Changes

- Removed managedGroups and managedDepartments from plan-level authorization
- Moved leader-level authorization into leader object inside plan-level authorization. This will allow authorization of learners in the future
- Added canViewLicenseAllocations and canViewUserSkillLevels as user-level authorization instead of plan level
- Removed canConnectRepositories

## Track Last Plan Accessed for Leader

### Request Details

Given a leader's `userHandle`, this endpoint tracks and returns a list of plans a leader has access to as well as a last accessed plan. When passing a `planId`, it is stored as the

new last accessed plan for the leader and returned as the current plan. This endpoint is to be used in conjunction with the [AccountSwitcherLite component](#).

**HTTP Method:** POST

**URI Template:**

- Staging: <https://plans-api-stage.vnerd.com/v1/users/plans>
- Production: <https://plans-api.vnerd.com/v1/users/plans>

**Headers:**

- Authorization: Token <value>
- You'll need to get a token to use this API. Talk to @dark-matter-dev in #dev-plan-management to get one.

## Request Payload

```
{
  "data": {
    "userHandle": "user-handle",
    "planId": "plan-id"
  }
}
or
{
  "data": {
    "userHandle": "user-handle"
  }
}
```

## Result

Successful Response (200):

```
{
  "data": {
```

```
"currentPlan": {
  planId: "plan-id",
  displayName: "display-name"
},
"plans": [
  {
    planId: "plan-id",
    displayName: "display-name"
    totalLicenses: 4
  }
]
}
```

Unprocessable Entity Response (422):

```
{
  "errors": [
    {
      "detail": "<Detail message here>",
      "status": "422",
      "code": "missing-required-fields"
    }
  ]
}
```

Invalid Token Response (401):

```
{
  "errors": [
    {
      "detail": "<Detail message here>",
      "status": "401",
      "code": "invalid-token"
    }
  ]
}
```

## Notes

- The `userHandle` is required and the `planId` is optional.

## Changes

- None

## Remove user from Plan

## Request Details

Fully remove member from plan.

**HTTP Method:** DELETE

### URI Template:

- Staging:  
<https://plans-api-stage.vnerd.com/v4/plans/{:planId}/users/{:userHandle}>
- Production: <https://plans-api.vnerd.com/v4/plans/{:planId}/users/{:userHandle}>

### Headers:

- Authorization: Token <value>

## Token authorization

You'll need to get a token to use this API. Talk to @dark-matter-dev in 

<https://pluralsight.slack.com/archives/C8ZCWDZ5H> - Connect your Slack account to get one.

Put your token in an authorization header: Authorization: Token <value>

# Response

Successful No Content (204) Response:

```
{ }
```

Not Found (404) Response:

```
{
  "errors": [
    {
      "detail": "<Detail message here>",
      "status": "404",
      "code": "plan-not-found"
    },
    {
      "detail": "<Detail message here>",
      "status": "404",
      "code": "member-not-found"
    }
  ]
}
```

## Member (alias: Learner) API Endpoints

- [Get All Members \(alias: Learners\) for a Plan](#)
- [Get Member \(alias: Learner\) Details](#)
- [Update Member \(alias: Learner\) Note](#)
- [Update Member \(alias: Learner\) Products](#)
- [Unassign Member \(alias: Learner\)](#)
- [Remove Member \(alias: Learner\) From Expired Plan](#)

## Get All Members (alias: Learners) for a Plan

## Request Details

Given a planId, returns all learners for that plan, or a 404 if the plan is not found.

To query for a single member, use our [Get learner details](#) endpoint.

**HTTP Method:** GET

**URI Template:**

- Staging: <https://plans-api-stage.vnerd.com/v4/plans/:planId/members>
- Production: <https://plans-api.vnerd.com/v4/plans/:planId/members>

**Headers:**

- Authorization: Token
- You'll need to get a token to use this API. Talk to @dark-matter-dev in <https://pluralsight.slack.com/archives/C8ZCWDZ5H> to get one.

Response

```
{
  "data": [
    {
      "id": "uy9a1d08-b9e5-4610-b6c9-c25103e37852",
      "planId": "1800contacts",
      "teamIds": [ "37211c4d-de61-4498-9c84-7486322876b1" ],
      "userHandle": "0b9a1d08-b9e5-4610-b6c9-c25103e37598",
      "startDate": "2016-10-12T20:34:34.48+00:00",
      "note": "some-note"
    },
    {
      "id": "uy9a1d08-b9e5-4610-b6c9-c25103e37852",
      "planId": "1800contacts",
      "teamIds": [ "73eedaf-00e1-4550-9450-fa3b23f598fe",
        "37211c4d-de61-4498-9c84-7486322876b1" ],
      "userHandle": "108b8ae9-79f6-4eb7-bff4-bcd18218aaa5",
      "startDate": "2015-10-10T00:00:00+00:00",
      "note": "some-other-note"
    }
  ]
}
```

```
}
```

## Notes

teamIds is empty if the user is not assigned to a team.

## Get Member (alias: Learner) Details Request Details

Given a userHandle, returns the plan member information for that user, or a 404 if the user is not a member of a plan.


This can be used to self-heal the message [ps.plans.member-added-to-plan.v1](#)

**HTTP Method:** GET

**URI Template:**

- Staging: <https://plans-api-stage.vnerd.com/v4/members/:userHandle>
- Production: <https://plans-api.vnerd.com/v4/members/:userHandle>

**Headers:**

- Authorization: Token <value>
- You'll need to get a token to use this API. Talk to @dark-matter-dev in  <https://pluralsight.slack.com/archives/C8ZCWDZ5H> - Connect your Slack account to get one.

## Response

```
{  
  "data": {  
    "id": "6c4ad52b-cb02-4624-89da-c54e1ca8af59",
```

```
"userHandle": "user-handle",
"planId": "pluralsight-engineering-1d36a",
"teamIds": ["52f1611d-552c-4f57-a5ca-8d9b7e0c29f5"],
"note": "some-note",
"startDate": "2015-12-02T17:02:10.356+00:00",
"products": [
{
  "id": "ee4e7c4c-55f2-4bed-9511-5138eeda88bc",
  "optionId": "f6cb46c6-5ccc-472b-9779-4c4c7fd1096d"
}
]
}
```

### Notes

- teamIds is empty if the user is not assigned to a team
- teamIds will only include the teams the user is directly assigned to

## Changes

### Breaking changes from **v3**:

- changed teamId to teamIds, which allows for a member to be assigned to multiple teams

## Update Member (alias: Learner) Note

### Request Details

Change the note of an existing member on a plan.

**HTTP Method:** PUT

**URI Template:**



- Staging:  
<https://plans-api-stage.vnerd.com/v4/plans/{:planId}/members/{:userHandle}/notes>
- Production:  
<https://plans-api.vnerd.com/v4/plans/{:planId}/members/{:userHandle}/notes>

### Headers:

- Authorization: Token <value>

### Request Payload

```
{
  "data": {
    "note": "New note"
  }
}
```

- **Note:** New note for member (can be null to unset a note)

## Token authorization

You'll need to get a token to use this API. Talk to @dark-matter-dev in 

<https://pluralsight.slack.com/archives/C8ZCWDZ5H> - Connect your Slack account to get one.

Put your token in an authorization header: Authorization: Token <value>

## Response

Successful No Content (204) Response:

```
{ }
```

Not Found (404) Response:

```
{
  "errors": [
    {
      "detail": "<Detail message here>",
      "status": "404",
      "code": "plan-not-found"
    },
    {
      "detail": "<Detail message here>",
      "status": "404",
      "code": "member-not-found"
    }
  ]
}
```

Bad Request (400) Response:

```
{
  "errors": [
    {
      "detail": "<Detail message here>",
      "status": "400",
      "code": "invalid-note"
    },
    {
      "detail": "<Detail message here>",
      "status": "400",
      "code": "unknown-error-occurred"
    }
  ]
}
```

Unprocessable Entity (422) Response:

```
{
  "errors": [
    {
      "detail": "<Detail message here>",
```

```
"status": "422",  
"code": "missing-required-fields"  
}  
]  
}
```

## Update Member (alias: Learner) Products

### Request Details

Update member products.

**HTTP Method:** POST

**URI Template:**

- Staging:  
<https://plans-api-stage.vnerd.com/v4/plans/{:planId}/members/{:userHandle}/products>
- Production:  
<https://plans-api.vnerd.com/v4/plans/{:planId}/members/{:userHandle}/products>

**Headers:**

- Authorization: Token <value>

### Request Payload

```
{  
  "data": {  
    "productIdsToAssign": [ "a0c07d8e-4169-47c0-b7fa-d4815924b679" ],  
    "productIdsToRevoke": [ "a3c882ea-4966-4b3d-bd2d-15310caa46f9" ]  
  }  
}
```

- productIdsToAssign: Product IDs to assign to member. All IDs passed in will be added current set of products for the member.

- `productIdsToRevoke`: Product IDs to remove from member

## Notes

- Either `productIdsToAssign` or `productIdsToRevoke` must be non-empty or an error will be returned.
- If the user is a leader on the plan, but not a member, assigning products via this API will add the user as a learner on the plan.
- If all products are removed from a member, the user will enter an unlicensed state but remain assigned to the plan.

## Token authorization

You'll need to get a token to use this API. Talk to @dark-matter-dev in 

<https://pluralsight.slack.com/archives/C8ZCWDZ5H> - Connect your Slack account to get one.

Put your token in an authorization header: `Authorization: Token <value>`

## Response

Successful No Content (204) Response:

```
{ }
```

Unprocessable Entity (422) Response:

```
{
  "errors": [
    {
      "detail": "<Detail message here>",
      "status": "422",
      "code": "no-learner-base-product"
    }
  ]
}
```

```
},
{
  "detail": "<Detail message here>",
  "status": "422",
  "code": "not-enough-licenses"
},
{
  "detail": "<Detail message here>",
  "status": "422",
  "code": "existing-member-other-account"
},
{
  "detail": "<Detail message here>",
  "status": "422",
  "code": "user-has-individual-subscription"
},
]
}
```

Not Found (404) Response:

```
{
  "errors": [
    {
      "detail": "<Detail message here>",
      "status": "404",
      "code": "plan-not-found"
    },
    {
      "detail": "<Detail message here>",
      "status": "404",
      "code": "member-not-found"
    }
  ]
}
```

Bad Request (400) Response:

```
{
  "errors": [
    {
      "detail": "<Detail message here>",
      "status": "400",
      "code": "product-required"
    },
    {
      "detail": "<Detail message here>",
      "status": "400",
      "code": "invalid-product-id"
    },
    {
      "detail": "<Detail message here>",
      "status": "400",
      "code": "unknown-error-occurred"
    }
  ]
}
```

## Changes

- A status 422 with error code `all-learner-products-revoked` used to be returned when attempting to remove all products from a learner. Attempting this now succeeds and results in the user becoming an unlicensed learner.

## Unassign Member (alias: Learner)

### Request Details

Remove a member's license(s) from plan.

NOTE: This endpoint will remove the licenses from a user on a plan, but it will no longer remove the user from the plan. This can result in the user being an unlicensed Learner state. To fully remove a user from a plan, use [this endpoint](#).

**HTTP Method:** DELETE

**URI Template:**

- Staging:  
<https://plans-api-stage.vnerd.com/v4/plans/{:planId}/members/{:userHandle}>
- Production:  
<https://plans-api.vnerd.com/v4/plans/{:planId}/members/{:userHandle}>

**Headers:**

- Authorization: Token <value>

## Token authorization

You'll need to get a token to use this API. Talk to @dark-matter-dev in 

<https://pluralsight.slack.com/archives/C8ZCWDZ5H> - Connect your Slack account to get one.

Put your token in an authorization header: Authorization: Token <value>

## Response

Successful No Content (204) Response:

Not Found (404) Response:

```
{
  "errors": [
    {
```

```
"detail": "<Detail message here>",
"status": "404",
"code": "plan-not-found"
},
{
"detail": "<Detail message here>",
"status": "404",
"code": "member-not-found"
}
]
}
```

## Changes

- 06/2023 - No longer fully removes user association with a plan, which can result in unlicensed learners

## Remove Member (alias: Learner) From Expired Plan

### Request Details

Given a `userHandle`, removes the user from the expired plan they are currently on. This is useful if the user is trying to buy another subscription or redeem an offer code.


**HTTP Method:** DELETE

**URI Template:**

- Staging: <https://plans-api-stage.vnerd.com/v4/members/:userHandle/expired>
- Production: <https://plans-api.vnerd.com/v4/members/:userHandle/expired>

**Headers:**



- Authorization: Token <value>
- You'll need to get a token to use this API. Talk to @dark-matter-dev in   
<https://pluralsight.slack.com/archives/C8ZCWDZ5H> - Connect your Slack account to get one.

## Responses

The API will return a status code 204 if the user is removed successfully. If the `userHandle` does not exist, a status code of 404 will be returned. If the plan is not expired, a status code of 400 will be returned.

## Changes

- None

## Team API Endpoints

Create Team  
Rename Team  
Change Team Description  
Change Team Parent  
Dissolve Team  
Team Member (alias: Learner) Endpoints  
Team Manager Endpoints

## Create Team

## Request Details

Create a team on a given plan.

**HTTP Method:** POST

**URI Template:**

- Staging: <https://plans-api-stage.vnerd.com/v4/plans/{:planId}/teams>
- Production: <https://plans-api.vnerd.com/v4/plans/{:planId}/teams>

**Headers:**

- Authorization: Token <value>

## Request Payload

```
{
  "data": {
    "name": "Test Team",
    "description": "This is a team description.",
    "parentTeamId": "b3fa8014-3fd3-4f71-94f3-eb7c4c7c2376"
  }
}
```

Note: Both Name & Description have a limit of 200 characters

- name: Team's name REQUIRED
- description: Team description OPTIONAL
- parentTeamId: Team's parent team OPTIONAL – If not passed, it will not assign team to parent (creates unattached team).

## Token authorization

You'll need to get a token to use this API. Talk to @dark-matter-dev in 

<https://pluralsight.slack.com/archives/C8ZCWDZ5H> - Connect your Slack account to get one.

Put your token in an authorization header: Authorization: Token <value>

# Response

Successful Created (201) Response:

```
{
  "data": {
    "id": "c6fg8564-3fd3-4f71-94f3-gb7c4c7c7423",
    "name": "Test Team",
    "parentTeamId": "b3fa8014-3fd3-4f71-94f3-eb7c4c7c2376",
    "description": "This is a team description."
  }
}
```

Not Found (404) Response:

```
{
  "errors": [
    {
      "detail": "<Detail message here>",
      "status": "404",
      "code": "plan-not-found"
    }
  ]
}
```

Unprocessable Entity (422) Response:

```
{
  "errors": [
    {
      "detail": "<Detail message here>",
      "status": "422",
      "code": "missing-required-fields"
    },
    {
      "detail": "<Detail message here>",
      "status": "422",
      "code": "invalid-team-name"
    }
  ]
}
```

```
},
{
  "detail": "<Detail message here>",
  "status": "422",
  "code": "invalid-description"
},
{
  "detail": "<Detail message here>",
  "status": "422",
  "code": "team-already-exists"
},
{
  "detail": "<Detail message here>",
  "status": "422",
  "code": "feature-not-supported"
}
]
}
```

## Changes

- Replaced groupId with parentTeamId. Teams can now be nested under other teams. Note that if the plan is not part of the experience supporting nested teams, the endpoint will return a 422 error with code feature-not-supported if a parent team is provided.

## Rename Team

## Request Details

Rename a team on a given plan.

**HTTP Method:** PUT

## URI Template:

- Staging:  
<https://plans-api-stage.vnerd.com/v4/plans/{:planId}/teams/{:teamId}/names>
- Production:  
<https://plans-api.vnerd.com/v4/plans/{:planId}/teams/{:teamId}/names>

## Headers:

- Authorization: Token <value>

## Request Payload

```
{  
  "data": {  
    "name": "New Team Name"  
  }  
}
```

- **Name:** Team's name (**Required**)

**Note:** *This field has a limit of 200 characters*

# Token authorization

You'll need to get a token to use this API. Talk to @dark-matter-dev in 

<https://pluralsight.slack.com/archives/C8ZCWDZ5H> - Connect your Slack account to get one.

Put your token in an authorization header: Authorization: Token <value>

# Response

Successful No Content (204) Response:

```
{ }
```

Not Found (404) Response:

```
{
  "errors": [
    {
      "detail": "<Detail message here>",
      "status": "404",
      "code": "plan-not-found"
    },
    {
      "detail": "<Detail message here>",
      "status": "404",
      "code": "team-not-found"
    }
  ]
}
```

Unprocessable Entity (422) Response:

```
{
  "errors": [
    {
      "detail": "<Detail message here>",
      "status": "422",
      "code": "missing-required-fields"
    },
    {
      "detail": "<Detail message here>",
      "status": "422",
      "code": "invalid-team-name"
    }
  ]
}
```

Conflict (409) Response:

```
{
```

```
"errors": [  
  {  
    "detail": "<Detail message here>",  
    "status": "409",  
    "code": "team-already-exists"  
  }  
]
```

## Changes

- None

## Change Team Description Request Details

Change a team's description.

**HTTP Method:** PUT

**URI Template:**

- Staging:  
<https://plans-api-stage.vnerd.com/v4/plans/{:planId}/teams/{:teamId}/descriptions>
- Production:  
<https://plans-api.vnerd.com/v4/plans/{:planId}/teams/{:teamId}/descriptions>

**Headers:**

- Authorization: Token <value>

## Request Payload

```
{
```

```
"data": {  
  "description": "Some new description for the team."  
}
```

- **Description:** Team's description (**Required**)

**Note:** *This field has a limit of 200 characters*

## Token authorization

You'll need to get a token to use this API. Talk to @dark-matter-dev in 

<https://pluralsight.slack.com/archives/C8ZCWDZ5H> - Connect your Slack account to get one.

Put your token in an authorization header: Authorization: Token <value>

## Response

Successful No Content (204) Response:

```
{ }
```

Not Found (404) Response:

```
{  
  "errors": [  
    {  
      "detail": "<Detail message here>",  
      "status": "404",  
      "code": "plan-not-found"  
    },  
    {  
      "detail": "<Detail message here>",  
      "status": "404",
```



```
"code": "team-not-found"
}
]
}
```

Unprocessable Entity (422) Response:

```
{
  "errors": [
    {
      "detail": "<Detail message here>",
      "status": "422",
      "code": "missing-required-fields"
    },
    {
      "detail": "<Detail message here>",
      "status": "422",
      "code": "invalid-description"
    }
  ]
}
```

## Change Team Parent

### Request Details

Move team to a new parent team or detach from parent.

**HTTP Method:** PUT

**URI Template:**

- Staging:  
<https://plans-api-stage.vnerd.com/v4/plans/{:planId}/teams/{:teamId}/parents>
- Production:  
<https://plans-api.vnerd.com/v4/plans/{:planId}/teams/{:teamId}/parents>

## Headers:

- Authorization: Token <value>

## Request Payload

### Attach to parent team

```
{
  "data": {
    "parentTeamId": "b3fa8014-3fd3-4f71-94f3-eb7c4c7c2376"
  }
}
```

### Detach from parent team

```
{
  "data": {
    "parentTeamId": null
  }
}
```

- **ParentTeamId:** Team's parent team REQUIRED – Can be null if wanting to detach from parent

## Note

If the plan is not part of the experience supporting nested teams, the endpoint will return a 422 error with code feature-not-supported if a parent team is provided

For V3 version supporting groups: [Plans API: Change team parent group](#)

## Token authorization

You'll need to get a token to use this API. Talk to @dark-matter-dev in 

<https://pluralsight.slack.com/archives/C8ZCWDZ5H> - Connect your Slack account to get one.

Put your token in an authorization header: Authorization: Token <value>

## Response

Successful No Content (204) Response:

```
{ }
```

Not Found (404) Response:

```
{
  "errors": [
    {
      "detail": "<Detail message here>",
      "status": "404",
      "code": "plan-not-found"
    },
    {
      "detail": "<Detail message here>",
      "status": "404",
      "code": "team-not-found"
    }
  ]
}
```

Unprocessable Entity (422) Response:

```
{
  "errors": [
    {
      "detail": "<Detail message here>",
      "status": "422",

```

```
"code": "missing-required-fields"
}
]
}
```

## Changes

- None

## Dissolve Team

### Request Details

Dissolve a team. This will remove all managers from the team and move any members under the team to be unassigned.

**HTTP Method:** DELETE

**URI Template:**

- Staging: <https://plans-api-stage.vnerd.com/v4/plans/{:planId}/teams/{:teamId}>
- Production: <https://plans-api.vnerd.com/v4/plans/{:planId}/teams/{:teamId}>

**Headers:**

- Authorization: Token <value>

## Token authorization

You'll need to get a token to use this API. Talk to @dark-matter-dev in 

<https://pluralsight.slack.com/archives/C8ZCWDZ5H> - Connect your Slack account to get one.

Put your token in an authorization header: Authorization: Token <value>

## Response

Successful No Content (204) Response:

```
{ }
```

Not Found (404) Response:

```
{  
  "errors": [  
    {  
      "detail": "<Detail message here>",  
      "status": "404",  
      "code": "plan-not-found"  
    },  
    {  
      "detail": "<Detail message here>",  
      "status": "404",  
      "code": "team-not-found"  
    }  
  ]  
}
```

## Team Member (alias: Learner) Endpoints

[Add Member \(alias: Learner\) to Team](#)

[Move Member \(alias: Learner\) to Team](#)

[Remove Member \(alias: Learner\) from Team](#)

### Add Member (alias: Learner) to Team

# Request Details

Add member to a team.

**HTTP Method:** PUT

**URI Template:**

- Staging:  
<https://plans-api-stage.vnerd.com/v4/plans/{:planId}/members/{:userHandle}/teams/{:teamId}>
- Production:  
<https://plans-api.vnerd.com/v4/plans/{:planId}/members/{:userHandle}/teams/{:teamId}>

**Headers:**

- Authorization: Token <value>

## Note

This endpoint will add the plan user to a new team each time the endpoint is called.

## Token authorization

You'll need to get a token to use this API. Talk to @dark-matter-dev in 

<https://pluralsight.slack.com/archives/C8ZCWDZ5H> - Connect your Slack account to get one.

Put your token in an authorization header: Authorization: Token <value>

## Response

Successful No Content (204) Response:

```
{ }
```

Not Found (404) Response:

```
{
  "errors": [
    {
      "detail": "<Detail message here>",
      "status": "404",
      "code": "plan-not-found"
    },
    {
      "detail": "<Detail message here>",
      "status": "404",
      "code": "user-not-found"
    },
    {
      "detail": "<Detail message here>",
      "status": "404",
      "code": "team-not-found"
    }
  ]
}
```

Unprocessable Entity (422) Response:

```
{
  "errors": [
    {
      "detail": "<Detail message here>",
      "status": "422",
      "code": "teams-without-learner"
    }
  ]
}
```

# Move Member (alias: Learner) to Team

## Request Details

Move member to new teams. This will remove member from all assigned teams and move to the teams provided. If no team is provided, the member will be removed from all teams and moved to unassigned.

**HTTP Method:** PUT

**URI Template:**

- Staging:  
<https://plans-api-stage.vnerd.com/v4/plans/{:planId}/members/{:userHandle}/teams>
- Production:  
<https://plans-api.vnerd.com/v4/plans/{:planId}/members/{:userHandle}/teams>

**Headers:**

- Authorization: Token <value>

## Request Payload

```
{
  "data": {
    "teamIds": [ "b3fa8014-3fd3-4f71-94f3-eb7c4c7c2376" ]
  }
}
{
  "data": {
    "teamIds": [ ]
  }
}
```



- **TeamIds:** Team IDs to add member to. Team IDs must be supplied. If the list is empty, or if the value is null, the member will be unassigned from all teams.  
REQUIRED

## Token authorization

You'll need to get a token to use this API. Talk to @dark-matter-dev in 

<https://pluralsight.slack.com/archives/C8ZCWDZ5H> - Connect your Slack account to get one.

Put your token in an authorization header: Authorization: Token <value>

## Response

Successful No Content (204) Response:

```
{ }
```

Not Found (404) Response:

```
{
  "errors": [
    {
      "detail": "<Detail message here>",
      "status": "404",
      "code": "plan-not-found"
    },
    {
      "detail": "<Detail message here>",
      "status": "404",
      "code": "member-not-found"
    },
    {
      "detail": "<Detail message here>",
```

```
"status": "404",  
"code": "team-not-found"  
}  
]  
}
```

Unprocessable Entity (422) Response:

```
{  
  "errors": [  
    {  
      "detail": "<Detail message here>",  
      "status": "422",  
      "code": "missing-required-fields"  
    },  
    {  
      "detail": "<Detail message here>",  
      "status": "422",  
      "code": "teams-without-learner"  
    }  
  ]  
}
```

## Remove Member (alias: Learner) from Team

### Request Details

Remove member from a team. Will move the member to unassigned if member has been removed from all teams.

**HTTP Method:** DELETE

**URI Template:**

- Staging:  
<https://plans-api-stage.vnerd.com/v4/plans/{:planId}/members/{:userHandle}/teams/{:teamId}>
- Production:  
<https://plans-api.vnerd.com/v4/plans/{:planId}/members/{:userHandle}/teams/{:teamId}>

## Headers:

- Authorization: Token <value>

# Token authorization

You'll need to get a token to use this API. Talk to @dark-matter-dev in 

<https://pluralsight.slack.com/archives/C8ZCWDZ5H> - Connect your Slack account to get one.

Put your token in an authorization header: Authorization: Token <value>

# Response

Successful No Content (204) Response:

```
{ }
```

Not Found (404) Response:

```
{
  "errors": [
    {
      "detail": "<Detail message here>",
      "status": "404",
      "code": "plan-not-found"
    },
  ],
}
```

```
"detail": "<Detail message here>",
"status": "404",
"code": "member-not-found"
},
{
"detail": "<Detail message here>",
"status": "404",
"code": "team-not-found"
}
]
}
```

## Team Manager Endpoints

[Add Team Manager \(From Existing User\)](#)

[Remove Team Manager](#)

Add label

## Invite Endpoints

[Invite Member \(alias: Learner\)](#)

[Invite Team Manager](#)

[Update Invite Note](#)

[Cancel Invite](#)

Add label

## Invite Member (alias: Learner)

## Request Details

Create an invite for a member.

**Note:** This endpoint does not allow re-sending invites, but *is for new invites only*. If an invite has expired and this invite is called it will send another, by creating a new one. If an active invite is passed in, it will not resend a new invite, but will just return a status of OK.

**HTTP Method:** POST

**URI Template:**

- Staging: <https://plans-api-stage.vnerd.com/v4/plans/{:planId}/invites/members>
- Production: <https://plans-api.vnerd.com/v4/plans/{:planId}/invites/members>

**Headers:**

- Authorization: Token <value>

## Request Payload

```
{
  "data": {
    "email": "some-user@example.com",
    "teamIds": [ "b3fa8014-3fd3-4f71-94f3-eb7c4c7c2376" ],
    "note": "Some member note",
    "productIds": [ "a0c07d8e-4169-47c0-b7fa-d4815924b679" ]
  }
}
```

- email: Email of member to invite REQUIRED
- teamIds: Team IDs to invite member to. OPTIONAL -- Default is unassigned
- note: A note for the member invite OPTIONAL
- productIds: Product IDs to assign to member OPTIONAL

## Token authorization

You'll need to get a token to use this API. Talk to @dark-matter-dev in 

<https://pluralsight.slack.com/archives/C8ZCWDZ5H> - Connect your Slack account to get one.

Put your token in an authorization header: Authorization: Token <value>

## Response

Successful Created (201) Response:

```
{
  "data": {
    "id": "b3fa8014-3fd3-4f71-94f3-eb7c4c7c2376",
    "email": "some-user@example.com",
    "teamIds": ["b3fa8014-3fd3-4f71-94f3-eb7c4c7c2376"],
    "note": "Some member note",
    "inviteRedemptionLink":
      "https://app-stage.pluralsight.com/id/createaccount/business?firstName=&lastName=&companyEmail=test%40example.net&redirectTo=https%3a%2f%2fapp-stage.pluralsight.com%2fplans-data%2finvites%2f1800contacts%2s4frsf6cf-a06e-43ae-84b1-82721ced0af0"
  }
}
```

Not Found (404) Response:

```
{
  "errors": [
    {
      "detail": "<Detail message here>",
      "status": "404",
      "code": "plan-not-found"
    },
    {
      "detail": "<Detail message here>",
```

```
"status": "404",  
"code": "team-not-found"  
}  
]  
}
```

Bad Request (400) Response:

```
{  
  "errors": [  
    {  
      "detail": "<Detail message here>",  
      "status": "400",  
      "code": "unknown-error-occurred"  
    }  
  ]  
}
```

Unprocessable Entity (422) Response:

```
{  
  "errors": [  
    {  
      "detail": "<Detail message here>",  
      "status": "422",  
      "code": "missing-required-fields"  
    },  
    {  
      "detail": "<Detail message here>",  
      "status": "422",  
      "code": "email-is-required"  
    },  
    {  
      "detail": "<Detail message here>",  
      "status": "422",  
      "code": "invalid-note"  
    },  
    {
```

```
"detail": "<Detail message here>",
"status": "422",
"code": "not-enough-licenses"
},
{
"detail": "<Detail message here>",
"status": "422",
"code": "feature-not-supported"
}
]
}
```

## Changes

- Replaced teamId with teamIds. Note that if the plan is not part of the experience supporting plan users on multiple teams, the endpoint will return a 422 error with code feature-not-supported if more than one team is provided.

## Invite Team Manager Request Details

Invite team managers to a team.

**HTTP Method:** POST

**URI Template:**

- Staging:  
<https://plans-api-stage.vnerd.com/v4/plans/{:planId}/invites/managers/teams/{:teamId}>
- Production:  
<https://plans-api.vnerd.com/v4/plans/{:planId}/invites/managers/teams/{:teamId}>

**Headers:**



- Authorization: Token <value>

## Request Payload

```
{
  "data": {
    "email": "manager@example.com",
    "permissionSetId": "1d5f1a5a-1795-46f1-bc5b-9667357a1be0",
    "shouldConsumeLicense": true
  }
}
```

- email: Manager's email REQUIRED
- permissionSetId: The ID of a permission set to assign the manager REQUIRED -- GUID
- shouldConsumeLicense: Specifies whether to also assign the manager a license OPTIONAL -- Default is false

## Token authorization

You'll need to get a token to use this API. Talk to @dark-matter-dev in REQUIRED to get one.

Put your token in an authorization header: Authorization: Token <value>

## Response

Successful Created (201) Response:

```
{
  "data": [
    {
      "id": "b3fa8014-3fd3-4f71-94f3-eb7c4c7c2376",
      "email": "manager@example.com",
```

```
"sendDate": "2019-10-06T07:00:00+00:00",
"inviteRedemptionLink":
"https://app-stage.pluralsight.com/id/createaccount/business?fir
stName=&lastName=&companyEmail=manager%40example.com&redirectTo=
https%3a%2f%2fapp-stage.pluralsight.com%2fplans-data%2finvites%2
f1800contacts%2s4frsf6cf-a06e-43ae-84b1-82721ced0af0"
}
]
}
```

Not Found (404) Response:

```
{
"errors": [
{
"detail": "<Detail message here>",
"status": "404",
"code": "plan-not-found"
},
{
"detail": "<Detail message here>",
"status": "404",
"code": "team-not-found"
},
{
"detail": "<Detail message here>",
"status": "404",
"code": "permission-set-id-not-found"
}
]
}
```

Unprocessable Entity (422) Response:

```
{
"errors": [
{
"detail": "<Detail message here>",
```

```
"status": "422",
"code": "missing-required-fields"
},
{
"detail": "<Detail message here>",
"status": "422",
"code": "invalid-permission-set-id"
},
{
"detail": "<Detail message here>",
"status": "422",
"code": "invalid-email"
},
{
"detail": "<Detail message here>",
"status": "422",
"code": "not-enough-licenses"
}
]
}
```

## Changes

- `PermissionSetId` is now passed in instead of `PermissionLevel`. The `permissionSetId` must be a manager permission set id.

## Update Invite Note

### Request Details

Change the note of an existing invitation.

**HTTP Method:** PUT

### URI Template:

- Staging:  
<https://plans-api-stage.vnerd.com/v4/plans/{:planId}/invites/{:inviteId}/note>
- Production:  
<https://plans-api.vnerd.com/v4/plans/{:planId}/invites/{:inviteId}/note>

### Headers:

- Authorization: Token <value>

### Request Payload

```
{  
  "data": {  
    "note": "New note"  
  }  
}
```

- **Note:** New note for invite (can be null to unset a note)

## Token authorization

You'll need to get a token to use this API. Talk to @dark-matter-dev in 

<https://pluralsight.slack.com/archives/C8ZCWDZ5H> - Connect your Slack account to get one.

Put your token in an authorization header: Authorization: Token <value>

## Response

Successful No Content (204) Response:

```
{ }
```

Not Found (404) Response:

```
{
  "errors": [
    {
      "detail": "<Detail message here>",
      "status": "404",
      "code": "plan-not-found"
    },
    {
      "detail": "<Detail message here>",
      "status": "404",
      "code": "invite-not-found"
    }
  ]
}
```

Bad Request (400) Response:

```
{
  "errors": [
    {
      "detail": "<Detail message here>",
      "status": "400",
      "code": "invalid-note"
    },
    {
      "detail": "<Detail message here>",
      "status": "400",
      "code": "unknown-error-occurred"
    }
  ]
}
```

## Changes

- None

# Cancel Invite

## Request Details

Cancels an invite.

**HTTP Method:** DELETE

**URI Template:**

- Staging: <https://plans-api-stage.vnerd.com/v4/plans/{:planId}/invites/{:inviteId}>
- Production: <https://plans-api.vnerd.com/v4/plans/{:planId}/invites/{:inviteId}>

**Headers:**

- Authorization: Token <value>

## Token authorization

You'll need to get a token to use this API. Talk to @dark-matter-dev in 

<https://pluralsight.slack.com/archives/C8ZCWDZ5H> - Connect your Slack account to get one.

Put your token in an authorization header: Authorization: Token <value>

## Response

Successful No Content (204) Response:

```
{ }
```

Not Found (404) Response:

```
{  
  "errors": [  

```

```
{
  "detail": "<Detail message here>",
  "status": "404",
  "code": "plan-not-found"
},
{
  "detail": "<Detail message here>",
  "status": "404",
  "code": "invite-not-found"
}
]
```

## Plan Management RabbitMQ Queues

ps.plans.invite-canceled.v3  
ps.plans.invite-created.v4  
ps.plans.invite-redeemed.v4  
ps.plans.member-added-to-plan.v1  
ps.plans.member-assigned-to-team.v1  
ps.plans.member-note-updated.v1  
ps.plans.member-removed-from-plan.v1  
ps.plans.member-removed-from-team.v1  
ps.plans.plan-archived.v1  
ps.plans.plan-updated.v5  
ps.plans.team-manager-added.v4  
ps.plans.team-manager-removed.v3  
ps.plans.team-removed.v2  
ps.plans.team-updated.v3  
ps.plans.user-authorization-updated.v3

**ps.plans.invite-canceled.v3**

```
{
  "id": "a8895b9d-5677-42fc-903f-b37ce2171f8f",
  "planId": "plan-id",
  "messagePublishedAt": "2015-12-31T01:23:45+00:00"
}
```

Published when a member invite is canceled or resent. (A resend will issue a `ps.plans.invite-canceled.v3` and `ps.plans.invite-created.v4`)

#### Upgrading from previous versions:

- Deprecacted `inviteId` in v2 (called `deprecatedInviteId`), removed completely in v3.
- `id` is the new identifier for an invite. v2 contains both the old `deprecatedInviteId` and the new `id` if you need to migrate

## ps.plans.invite-created.v4

```
{
  "id": "a8895b9d-5677-42fc-903f-b37ce2171f8f",
  "planId": "plan-id",
  "email": "address@email.com",
  "expiresOn": "2016-01-30T01:23:45+00:00",
  "messagePublishedAt": "2015-12-31T01:23:45+00:00"
}
```

#### Upgrading from previous versions:

- Deprecacted `inviteId` in v3 (called `deprecatedInviteId`), removed completely in v4.
- `id` is the new identifier for an invite. v3 contains both the old `deprecatedInviteId` and the new `id` if you need to migrate
- Added dashes between words in the exchange name for v3 and beyond



## ps.plans.invite-redeemed.v4

```
{
  "id": "a8895b9d-5677-42fc-903f-b37ce2171f8f",
  "planId": "plan-id",
  "userHandle": "some-id",
  "messagePublishedAt": "2015-12-31T01:23:45+00:00"
}
```

### Upgrading from previous versions:

- Deprecacted `inviteId` in v3 (called `deprecatedInviteId`), removed completely in v4.
- `id` is the new identifier for an invite. v3 contains both the old `deprecatedInviteId` and the new `id` if you need to migrate
- Removed `firstName`, `lastName`, and `email` in v3. You should get these from Identity.
- Added dashes between words in the exchange name for v3 and beyond

## ps.plans.member-added-to-plan.v1



Owned by [Gemma Grover](#)

Last updated: [Feb 24, 2023](#) 1 min read 2 people viewed

Published when a member is added to a plan

```
{
  "id": "52956b89-c7ec-45de-973c-4dc633f0ef06",
  "planId": "some-plan-abcde",
  "userHandle": "user-handle",
  "startDate": "2015-12-31T01:23:43+00:00",
  "products": [
    {
      "productId": "ee4e7c4c-55f2-4bed-9511-5138eeda88bc",

```

```
"productOptionId": "f6cb46c6-5ccc-472b-9779-4c4c7fd1096d"
},
{
  "productId": "c2b5c2d0-6952-4973-9aa1-eacd821aa82d",
  "productOptionId": "4542eff2-fedc-4e6f-b3f9-10004e1a73e8"
}
],
"note": "123",
"messagePublishedAt": "2015-12-31T01:23:45+00:00"
}
```

**Breaking changes from `ps.plans.team-member-added.v3`:**

- Removed `ps.plans.team-member-moved.v2` event. The notion of moving members will not exist in the new experience. Members are just added and removed from teams
- Removed `teamId`, `firstName`, `lastName`, and `email`. Team relationships can be obtained from `ps.plans.member-assigned-to-team.v1` events, and user metadata should be obtained from identity BC events

**Related messages:**

- [ps.plans.member-removed-from-plan.v1](#) - published when a member is removed from a plan
- [ps.plans.member-assigned-to-team.v1](#) - published when a member is assigned to a team
- [ps.plans.member-removed-from-team.v1](#) - published when a member is removed from a team
- [ps.plans.member-note-updated.v1](#) - published when a member's note is updated
- `ps.identity.account-updated.v1` - published when a member's name or email is updated

## ps.plans.member-assigned-to-team.v1

Published when a member is assigned to a team

Plan users can be associated with multiple teams on a plan. A plan user can be added to multiple teams at the same time. This allows customers to represent matrical (matrix) hierarchies in their organization.

Teams can be nested underneath other teams. This replaces the concept of Departments and Groups. A team can have an optional parent team. Consumers should make no assumptions about the shape of the hierarchy (how many levels, etc.) and should assume unlimited nesting.

```
{
  "planId": "some-plan-abcde",
  "teamId": "887cb103-dd8f-446f-99c0-f63c1c327121",
  "userHandle": "user-handle",
  "isDirect": true,
  "messagePublishedAt": "2020-12-31T01:23:45+00:00"
}
```

## Notes

- This message, along with `ps.plans.member-removed-from-team.v1`, replaces `ps.plans.team-member-moved.v2`
- Members can be added to multiple teams. A `ps.plans.member-assigned-to-team.v1` event will be published for each team a member is added to. A member can also be associated with no teams (unassigned). In that case no `ps.plans.member-assigned-to-team.v1` events will be published for the member
- The event contains data for direct team relationships as well as nested relationships. A value of `true` for `isDirect` means the member is a direct child of the team. A value of `false` means the member is a descendant of the given team
- The `ps.plans.member-assigned-to-team.v1` can be published multiple times for the same (`planId`, `teamId`, `userHandle`) combination. This will happen when a team is moved and the `isDirect` field needs to be updated
- This event will only be published for members with an identity (not pending)

## Related messages:

- [ps.plans.member-added-to-plan.v1](#) - published when a member is added to a plan

- [ps.plans.member-removed-from-plan.v1](#) - published when a member is removed from a plan
- [ps.plans.member-removed-from-team.v1](#) - published when a member is removed from a team

## ps.plans.member-note-updated.v1

Published when a member's note is updated. This message is *not* published when the user is added, even if they are added with an initial note. Use [ps.plans.team-member-added.v3](#) to get the initial note, if any.

```
{
  "userHandle": "user-handle",
  "planId": "plan-id",
  "note": "123",
  "messagePublishedAt": "2015-12-31T01:23:45+00:00"
}
```

## ps.plans.member-removed-from-plan.v1



Owned by [Gemma Grover](#)

Last updated: [Feb 24, 2023](#) 1 min read 2 people viewed

Published when a member is removed from a plan

```
{
  "id": "52956b89-c7ec-45de-973c-4dc633f0ef06",
  "planId": "some-plan-abcde",
  "userHandle": "user-handle",
  "messagePublishedAt": "2015-12-31T01:23:45+00:00"
}
```

**Breaking changes from [ps.plans.team-member-removed.v3](#):**

- Removed `teamId`, `note`, and `startDate` fields

#### Related messages:

- [ps.plans.member-added-to-plan.v1](#) - published when a member is added to a plan
- [ps.plans.member-assigned-to-team.v1](#) - published when a member is assigned to a team
- [ps.plans.member-removed-from-team.v1](#) - published when a member is removed from a team

## ps.plans.member-removed-from-team.v1

Published when a member is removed from a team

Plan users can be associated with multiple teams on a plan. A plan user can be added to multiple teams at the same time. This allows customers to represent matrical (matrix) hierarchies in their organization.

Teams can be nested underneath other teams. This replaces the concept of Departments and Groups. A team can have an optional parent team. Consumers should make no assumptions about the shape of the hierarchy (how many levels, etc.) and should assume unlimited nesting.

```
{
  "planId": "some-plan-abcde",
  "teamId": "887cb103-dd8f-446f-99c0-f63c1c327121",
  "userHandle": "user-handle",
  "isDirect": true,
  "messagePublishedAt": "2020-12-31T01:23:45+00:00"
}
```

#### Notes

- This message, along with `ps.plans.member-assigned-to-team.v1`, replaces `ps.plans.team-member-moved.v2`

- Members can be removed from multiple teams. A `ps.plans.member-removed-from-team.v1` event will be published for each team a member is removed from. If a member has been removed from all teams they are associated with, they will become unassigned
- The event contains data for direct team relationships as well as nested relationships. A value of `true` for `isDirect` means the member is a direct child of the team. A value of `false` means the member is a descendant of the given team
- This event will only be published for members with an identity (not pending)

#### Related messages:

- `ps.plans.member-added-to-plan.v1` - published when a member is added to a plan
- `ps.plans.member-removed-from-plan.v1` - published when a member is removed from a plan
- `ps.plans.member-assigned-to-team.v1` - published when a member is assigned to a team

## ps.plans.plan-archived.v1

There are two instances when a plan can be archived:

1. When a plan is archived directly

```
{
  "planId": "1800contacts",
  "mergedIntoPlanId": null,
  "messagePublishedAt": "2015-12-31T01:23:45+00:00"
}
```

1. When a plan is merged into another plan

```
{
  "planId": "old-1800contacts",
  "mergedIntoPlanId": "1800contacts",
  "messagePublishedAt": "2015-12-31T01:23:45+00:00"
}
```

#### Related messages:

- `ps.plans.team-removed.v1` - published when a team is removed from a plan

- ps.plans.plan-administrator-removed.v3 - published when a plan administrator is removed from a plan
- ps.plans.team-manager-removed.v3 - published when a team manager is removed from a team
- ps.plans.team-member-removed.v3 - published when a member is removed from a plan

## ps.plans.plan-updated.v5

```
{
  "planId": "some-plan-abcde",
  "displayName": "Some plan",
  "createdOnUtc": "2015-01-01T00:00:00+00:00",
  "expiresOnUtc": "2020-07-07T00:00:00+00:00",
  "totalLicenseCount": 60,
  "isPilot": false,
  "features": {
    "advancedChannelAnalytics": false,
    "advancedSkillAnalytics": false,
    "advancedRoleIqAnalytics": false,
    "nonTechAnalytics": false,
    "roleIqAnalytics": false,
    "channelAnalytics": false,
    "analyticsUpsell": false,
    "qAndADownload": false,
    "transcender": false,
    "priorities": false,
    "projects": false,
    "customRoleIq": false,
    "skillsStrategy": false,
    "labs": false,
    "itil": false,
  },
  "messagePublishedAt": "2020-01-01T06:07:00z"
}
```

### **Breaking changes from** `ps.plans.plan-updated.v4`:

- Collapsed features and restrictions fields into just features and renamed some features to be more consistent
  - `nonTechAnalyticsRestricted` renamed to `nonTechAnalytics` and value inverted
  - `analyticsUpsellRestricted` renamed to `analyticsUpsell` and value inverted
  - `advancedSkillsReporting` renamed to `advancedSkillAnalytics`
  - `channelsReportingRestricted` renamed to `channelAnalytics` and value inverted
  - `advancedChannelsReporting` renamed to `advancedChannelAnalytics`
  - `roleIQAnalyticsRestricted` renamed to `roleIqAnalytics` and value inverted
  - `roleIQAnalytics` renamed to `advancedRoleIqAnalytics`
  - `gitPrimeAnalytics` removed
  - `mentoringEnabled` was removed since mentoring no longer exists
  - `codeSchoolEnabled` was removed since Code School no longer exists
- Removed features and moved them to user-level authorization data
  - `userSkillLevelRestricted`
  - `licenseAllocationSummaryRestricted`

## `ps.plans.team-manager-added.v4`

Published when a team manager is added to a plan

```
{  
  "id": "887cb103-dd8f-446f-99c0-f63c1c327121",  
  "planId": "some-plan-abcde",  
  "teamId": "dfd2a076-31eb-4e31-a7af-12ae28c1d81d",  
  "userHandle": "user-handle",  
  "messagePublishedAt": "2015-12-31T01:23:45+00:00"  
}
```

### **Breaking changes from** `ps.plans.team-manager-added.v3`:

- Removed `firstName`, `lastName`, and `email` which should be obtained from identity BC



- Groups and Departments are now considered Teams and adding managers at any level will utilize this message

#### **Related messages:**

- ps.plans.team-manager-removed.v4 - published when a team manager is removed from the plan
- ps.plans.team-updated.v3 - published when a team is created
- ps.identity.account-updated.v1 - published when a manager's name or email is updated

## ps.plans.team-manager-removed.v3

Published when a team manager is removed as the manager of a team

```
{  
  "id": "887cb103-dd8f-446f-99c0-f63c1c327121",  
  "planId": "some-plan-abcde",  
  "teamId": "dfd2a076-31eb-4e31-a7af-12ae28c1d81d",  
  "userHandle": "user-handle",  
  "messagePublishedAt": "2015-12-31T01:23:45+00:00"  
}
```

#### **Related messages:**

- ps.plans.team-manager-added.v4 - published when a team manager is added to a plan
- ps.plans.team-updated.v3 - published when a team is created

## ps.plans.team-removed.v2

Published when a team is removed from a plan

Teams can be nested underneath other teams. This replaces the concept of

Departments and Groups. A team can have an optional parent team. Consumers should

make no assumptions about the shape of the hierarchy (how many levels, etc.) and should assume unlimited nesting.

```
{
  "id": "887cb103-dd8f-446f-99c0-f63c1c327121",
  "planId": "some-plan-abcde",
  "messagePublishedAt": "2015-12-31T01:23:45+00:00"
}
```

### Changes

- Renamed teamId to id
- Groups and Departments are now considered Teams and will be removed through this message rather than ps.plans.group-removed.v1 and ps.plans.department-removed.v1

### Related messages:

- ps.plans.team-updated.v3

## ps.plans.team-updated.v3

Published when a team is created or when the name or parentTeamId is updated

Teams can be nested underneath other teams. This replaces the concept of Departments and Groups. A team can have an optional parent team. Consumers should make no assumptions about the shape of the hierarchy (how many levels, etc.) and should assume unlimited nesting.

```
{
  "id": "887cb103-dd8f-446f-99c0-f63c1c327121",
  "planId": "some-plan-abcde",
  "parentTeamId": "dfd2a076-31eb-4e31-a7af-12ae28c1d81d",
  "name": "Some Team",
  "description": "Description",
  "messagePublishedAt": "2015-12-31T01:23:45+00:00"
}
```

```
}
```

## Notes

- `parentTeamId` can be null

## Breaking changes from `ps.plans.team-updated.v2`:

- Renamed `teamId` to `id`
- Removed `groupId` field and replaced with `parentTeamId`
- Renamed `teamName` to `name`
- Groups and Departments are now considered Teams and will be updated through this message rather than `ps.plans.group-updated.v1` and `ps.plans.department-updated.v1`

## Related messages:

- `ps.plans.team-removed.v2` - published when a team is removed from a plan

# ps.plans.user-authorization-updated.v3

Published when an SDL user's authorization changes. Currently this is affected by being assigned permissions, manager status, and internal claims that grant access to all plans.

This message is focused on what a user has permission to do, not necessarily what role they have.

```
{
  "userHandle": "user-handle",
  "plans": [
    {
      "id": "some-plan-abcde",
      "leader": {
        "managedTeams": [
          {
```

```
"id": "887cb103-dd8f-446f-99c0-f63c1c327121"
}
],
"authz": {
  "canViewDashboard": true,
  "canViewAnalytics": true,
  "canViewAccount": true,
  "canViewBilling": true,
  "canViewPeople": true,
  "canViewPriorities": true,
  "canViewLogs": true,
  "canViewSettings": false,
  "canViewAllUsers": true,
  "canInviteUsers": true,
  "canViewCustomRoleIq": true,
  "canViewLicenseAllocations": true,
  "canViewUserSkillLevels": true,
  "canViewPrograms": true,
  "canViewCustomSandboxes": true,
  "canViewLearningPaths": true,
  "canViewSkillsAssessments": true,
  "canViewStudyGroups": true
}
}
}
],
"allPlansAuthz": {
  "canViewDashboard": false,
  "canViewAnalytics": false,
  "canViewAccount": false,
  "canViewBilling": false,
  "canViewPeople": false,
  "canViewPriorities": false,
  "canViewLogs": false,
  "canViewSettings": false,
```

```
"canViewAllUsers": false,
"canInviteUsers": false,
"canViewCustomRoleIq": false,
"canViewLicenseAllocations": false,
"canViewUserSkillLevels": false,
"canViewPrograms": false,
"canViewCustomSandboxes": false,
"canViewLearningPaths": false,
"canViewSkillsAssessments": false,
"canViewStudyGroups": false
},
"messagePublishedAt": "2021-10-02T00:00:00z",
"cacheUntil": "2021-10-02T00:00:00z"
}
```

### Changes

- Removed managedGroups and managedDepartments from plan-level authorization
- Moved leader-level authorization into leader object inside plan-level authorization. This will allow authorization of learners in the future
- Added canViewLicenseAllocations and canViewUserSkillLevels as user-level authorization instead of plan level
- Removed canConnectRepositories

### Related messages:

- ps.plans.team-manager-added.v4 - published when a team manager is added to a plan
- ps.plans.team-manager-removed.v4 - published when a team manager is removed from a plan
- ps.identity.user-claims-updated.v1 - published when internal claims change

## Plan Management DVS Topics

[Plan Management DVS V1 Topics](#)

## Plan Management DVS V1 Topics

[exp.plans.Manager](#)  
[exp.plans.v3.Member](#)  
[exp.plans.v2.MemberInvite](#)  
[exp.plans.v5.Membership](#)  
[exp.plans.v1.PermissionSet](#)  
[exp.plans.Plan](#)  
[exp.plans.v1.PlanMerge](#)  
[exp.plans.v1.PlanUser](#)  
[exp.plans.v1.PlanUserPermissionSet](#)  
[exp.plans.v1.PlanUserTeam](#)  
[exp.plans.v3.Team](#)

## Plan Management DVS V2 Topics and Consumers

### Migrated Topics (see migration guide [here](#))

[skills.plans.v1.Plan](#), consumers [here](#)

[skills.plans.v1.LearnerHistory](#), consumers [here](#)

[skills.plans.v1.PlanUserTeamHierarchy](#), consumers [here](#)

[skills.plans.v1.PlanUser](#), consumers [here](#)

[skills.plans.v1.Learner](#), consumers [here](#)

[skills.plans.v1.Team](#), consumers [here](#)

[skills.plans.v1.MergeHistory](#), consumers [here](#)

[skills.plans.v1.Leader](#), consumers [here](#)

[skills.plans.v1.LearnerInviteHistory](#), consumers [here](#)

[skills.plans.v1.PermissionSet](#), consumers [here](#)

## New-ish Topic Created Since Migration

[skills.plans.v1.PlanCloudMapping](#)  
(Proposed) [skills.plans.v1.PlanProductLicenseCounts](#)

## Documentation for the Plans-Org-Structure-Selection Package

```
import { TeamTree, Team } from
 '@pluralsight/plans-org-structure-selection'
import Button from '@pluralsight/ps-design-system-button'
function MyTeamComponent() {
  const teams = [
    {
      id: "pluralsight",
      name: "Pluralsight"
    },
    {
      id: "skills",
      name: "Skills",
      parentId: "pluralsight"
    },
    {
      id: "dark-matter",
      name: "Dark Matter",
      peopleCount: 6,
      parentId: "skills"
    }
  ];
  const onApply = (teams: Team[]) => {
```

```

console.log(`Selected teams: ${teams.map(team =>
team.name).join(', ')}`)
}
const trackEvent = (eventName: string, metadata?: unknown) => {
console.log('Event:', eventName, metadata)
}
const [hidden, setHidden] = useState(true)
return (
<TeamTree
teams={teams}
peopleWord="users"
trackEvent={trackEvent}
hidden={hidden}
onApply={onApply}
target={<Button onClick={() => setHidden(!hidden)}>Select
Teams</Button>}
/>
)
}

```

## TeamTree

Takes a list of teams and displays them in a tree structure. Built on top of `TreeFilter`.

Prop	Type	Required	Description
teams	Team[ ]	✓	List of teams to display



peopleWord	string	✓	Word that represents people. Ex: "users"
trackEvent	(eventName: string, metadata?: unknown)	✓	Called when an event is triggered. Used for tracking analytics
target	ReactNode   ({ ref: React.Dispatch<any> }) => React.ReactNode		The node (or render function) that the <a href="#">Popper</a> should be aligned to.
classes	TeamTreeClasses		CSS class(es) to apply to the TeamTree
itemProps	TreeItemProps		Default props for all TreeItems

<code>selectedIds</code>	<code>string[]</code>	IDs of teams that should be selected by default
<code>onApply</code>	<code>(teams: Team[]) =&gt; void</code>	Callback triggered to apply the selected teams via the Apply button. Passes in all of the selected teams.
<code>onSelect</code>	<code>(teams: Team[]) =&gt; void</code>	Called when a team is selected. Passes in all of the selected teams.
<code>placement</code>	<code>PopperJS.Placement</code>	If the popper is not being placed where you would expect, you can try changing this property.

<code>popperModifiers</code>	<code>PopperJS.Modifier[]</code>	Any additional custom modifiers. See <a href="https://popper.js.org/docs/v2/modifiers/">https://popper.js.org/docs/v2/modifiers/</a> for examples.
------------------------------	----------------------------------	--

## TreeFilter

Display and filter any tree data. Built on top of Tree.

Prop	Type	Required	Description
<code>nodes</code>	<code>Node[]</code>	✓	Nodes to display in the tree
<code>classes</code>	<code>TreeFilterClasses</code>		CSS classes
<code>itemProps</code>	<code>TreeItemProps</code>		Default props for all <code>TreeItems</code>

onCheck	(output: TreeOutput[], value: string, checked: boolean) => void	Called when a value is checked or unchecked
onToggle	(value: string, toggled: boolean) => void	Called when a section is toggled open or closed
defaultFilter	string	Default filter value
treeRef	React.Ref<any>	A reference to the Tree element

## Tree

Display any tree data.

Prop	Type	Required	Description
------	------	----------	-------------

nodes	TreeNode[]	✓	Nodes to display in the tree
classes	TreeClasses		CSS classes
itemProps	TreeItemProps		Default props for all TreeItems
onCheck	(output: TreeOutput[], value: string, checked: boolean) => void		Called when a value is checked or unchecked
onToggle	(value: string, toggled: boolean) => void		Called when a section is toggled open or closed

## Treeltem

Element displayed in a Tree

Prop	Type	Required	Description
label	string	✓	A display value for the item
value	string	✓	The value held by the item, e.g. id
checked	boolean		Whether or not the item is checked
children	ReactNode		React children of the TreeItem
classes	TreeItemClasses		CSS classes
disableSelection	boolean		Whether or not to disable selection of the team held by this TreeItem

<code>indeterminate</code>	<code>boolean</code>	If the <code>TreeItem</code> is neither explicitly checked nor unchecked
<code>metadata</code>	<code>any</code>	Metadata about the <code>TreeItem</code>
<code>open</code>	<code>boolean</code>	Whether or not to display any children
<code>Icon</code>	<code>ReactNode</code>	An optional icon to display with the <code>TreeItem</code>
<code>onCheck</code>	<code>(value: string, checked: boolean) =&gt; void</code>	Called when the node is checked or unchecked

onToggle	(value: string, toggled: boolean) => void	Called when the node is toggled open or closed
----------	---	--

## TreeContent

What a TreeItem is made of

Prop	Type	Required	Description
label	string	✓	Label of the item
classes	TreeContentClasses		CSS classes
metadata	string		Item(s) to display as metadata

## Tree Data Definitions

```
type TreeNode<TMetadata = unknown> = {
  id: string | number
  name: string
```



```

children?: TreeNode[]
state: TMetadata & Omit<TreeItemProps, 'label' | value'> & {
expanded? boolean
checked?: boolean
totalPeople?: boolean
}
}
type Team<TMetadata = unknown> = T & {
id: string
name: string
parentId?: string | null
peopleCount?: number
}

```

## CSS Class Types

```

type TeamTreeClasses = {
container?: string
componentContainer?: string
popperContainer?: string
TreeFilter?: TreeFilterClasses
}
type TreeFilterClasses = {
container?: string
textInput?: string
Tree?: TreeClasses
}
type TreeClasses = {
container?: string
TreeItem?: TreeItemClasses
}
type TreeItemClasses = {
caret?: string
content?: string
item?: string
}

```

```
noCaret?: string
selected?: string
checkbox?: string
TreeContent?: TreeContentClasses
}
type TreeContentClasses = {
content?: string
label?: string
metadata?: string
}
```

### **Dark Matter also known as People and Plan Management owns:**

- The [People](#) page and all tabs contained within. Admins, Managers and Pluralsight Employees see the people page
- The [Log](#) page. Only Admins and Pluralsight Employees can see the log page.
- The [Plan Directory page](#). Accessible only by Pluralsight Employees
- Some aspects of the [Settings](#) page. Accessible only by SOME Pluralsight Employees
- The Account Switcher is the white box shown in the screenshot below. Prism owns the navigation but we own the account switcher for admins and employees that 'own' multiple plans

### Page Navigation:

- [2022 OKRS](#)
- [JTBD](#)
- [Road Map](#)

# 2022 OKRS

Workboard:

<https://www.myworkboard.com/wb/goal/index?selectedMyGoalTab=2>

Original draft:

 2022 H1 Leader X OKRs

## JTBD

1. Organize and manage organizational hierarchy to support cohorted learning and analytics
2. Provide learners with access to content that enables them to skill up (professionally/personally)

## Road Map

### Delivery

Now - H1 2022

Next - H2 2022

Later - 2023 and  
beyond

Hub & Spoke ULA v1

[Org page Consolidation](#) - to consolidate license assignment and Team Manager actions to the Direcotry and Teams pages to support the new Pricing and Packaging work and ACG Integration in H2.

- License Dashboard added to Directory Page
- Allow Users to be assigned to Multiple Teams
- Custom invite message (supporting Onboarding teams work)
- Updated Invite Modal
- Updated Move Teams modal
- New Team Page to view Managers and Learners
- New Action Menu on Teams to manage Team Managers

### ACG Integration

- Unify Plan Management
- Migrate all users from ACG to Skills
- Enhancements to Leader flow within Skills to ACG

### Pricing and Packaging Re-architecture

- Flexible SKU assignment
- 1 learner to have multiple base licenses
- Automated Product Catalog

### AXELOS and vILT license assignment

- Consumption versus subscription license assignment

### Discovery

Now - H1 2022

Next - H2 2022

Later - 2023 and beyond

### Pricing and Packaging Re-architecture

- Flexible SKU assignment UX Testing
- Migration approach/plan

## ACG Integration

- cohorted migration discovery. How are we migrating plans over from ACG, timing surrounding this and communication plan

## License Pools

- License allocation/Pools at a Manager or Team level

## Learner/License access request

- Do learners want to request licenses to content their current subscription does not cover?
- Do Admins want to review request access to assign out unused licenses?

## Unlicensed Learners

- Allow access to learners to review free content and take Skills IQs for Managers and Admins to track and review.

## Team Hierarchy - adoption rate

- Assumption: Plans are not adopting the hierarchy structure because they don't have the tools to easily create
  - SSO Enablement/setup tooling
  - API's
  - User CSV

## How to continue to support automated plan management

- The above, but what LMS or HRIS Connectors are end user's looking for?
- How to know up front is Plan Management is handled through automation, like SSO and APIs
  - How do I change settings/fields associated with automation pieces

## Subscription Request Waitlist

- Allow people who are not current members to request access

Plan settings - Cross team settings for plan admins to control

- what are the settings Plan administrators want to control when initially setting up their plan
  - Communication preferences
  - manager preferences

Custom Permissions - manager permissions adoption rate

In-app usability

- CRUD actions
- 

Add label

## Plan Management (Dark Matter) Link Pack

All the links you'll need to bookmark!


## Team Operations


Team docs (front and back end coding standards found here):

[Plan Management \(Dark Matter\)](#)


Standup order: [Dark Matter Standup](#)

Plan Management Jira board:  [Plan Management](#)

Plan Management Help Center docs:  [Plan and team management | Pluralsight Help Center](#)

Merge Plans:  [Sign In | Pluralsight](#) (you must have employee-developer-sdl claim to view)

## Development

Plan Management Github:  <https://github.com/ps-dev/plan-management> - Connect your Github account

Skills Management Design System:

<https://ps-dev.github.io/skills-management-design-system/?path=/story/documentation-welcome--page>

General Pluralsight Design System:  [Home | Pluralsight Design System](#)

Issue with copying to AdHoc box:  [Fixed – Copy Paste not working in Remote Desktop Connection – Windows 10](#)

Estimation Sheet:  [Estimation](#)

Snyk Security Vulnerabilities: Use Okta Tile - Select Skills Organization - Search for Owned Repositories

Email Communication Templates: Use SparkPost Okta Tile - See template name by using  [Sign In | Pluralsight](#)

### Local


Local PS Identity Lite Login:

<https://app-local.pluralsight.com:3000/ps-identity-lite?redirectTo=https://app-local.pluralsight.com:3000/plans/adventureworks/people/org>

## Staging

Pluralsight environment:  [Plan Management](#)

Pluralsight admin user lookup: <https://app-stage.pluralsight.com/id/admin>

Pluralsight Communications Sent Products (where you can check to see if emails were sent):  [Stage Background Login | Pluralsight](#)

## Production

Pluralsight environment (login through Pluralsight App Okta tile):  [Plan Management](#)

# Deployment

Teamcity: <https://teamcity.vnerd.com/project/PlanManagement?mode=builds>

Octopus:

<https://octopus.vnerd.com/app#/Spaces-1/?projectId=ProjectGroups-34>

# Monitoring

\*Login credentials for Rabbit can be found in Lastpass. If the credentials don't autofill, click "edit" and copy and paste username and passwords.

**Dev portal Rabbit dead-letter shovel:** <https://dev-portal.vnerd.com/rabbitmq>

Choose env, then queue ie

*ps.subscriptions.business-products-state-updated.v2=>ps.plans.webevents*

Authorization Admin: <https://authorization.vnerd.com/admin2>



## Staging

Rabbit: <http://rabbit1-consume-staging.vnerd.com:15672/#/queues>

Application logs (Grumpy Bear):

[https://grumpybear.vnerd.com/app/kibana#/dashboard/18818140-d322-11e9-8342-e5bdf091ad9f?\\_g=h@44136fa&\\_a=h@b0fe9ee](https://grumpybear.vnerd.com/app/kibana#/dashboard/18818140-d322-11e9-8342-e5bdf091ad9f?_g=h@44136fa&_a=h@b0fe9ee)

Web logs:

[https://vpc-is-unified-weblogs-wjcx5paxf62yndsyq4nzm3smi.us-west-2.es.amazonaws.com/\\_plugin/kibana/app/kibana#/home?\\_g=\(\)](https://vpc-is-unified-weblogs-wjcx5paxf62yndsyq4nzm3smi.us-west-2.es.amazonaws.com/_plugin/kibana/app/kibana#/home?_g=())

AWS: Login via your AWS Okta tile

DVS Data Explorer: <https://dvs-staging.vnerd.com/dvs-data-explorer>

Product Catalog: <https://app-stage.pluralsight.com/product-catalog/products>

## Production

Rabbit: <http://rabbit1-consume-production.vnerd.com:15672/#/queues>

Application logs (Grumpy Bear):

[https://grumpybear.vnerd.com/app/kibana#/dashboard/59bdcab0-d322-11e9-8342-e5bdf091ad9f?\\_g=h@44136fa&\\_a=h@9edbce7](https://grumpybear.vnerd.com/app/kibana#/dashboard/59bdcab0-d322-11e9-8342-e5bdf091ad9f?_g=h@44136fa&_a=h@9edbce7)

Web logs:

[https://vpc-is-unified-weblogs-wjcx5paxf62yndsyq4nzm3smi.us-west-2.es.amazonaws.com/\\_plugin/kibana/app/kibana#/home?\\_g=\(\)](https://vpc-is-unified-weblogs-wjcx5paxf62yndsyq4nzm3smi.us-west-2.es.amazonaws.com/_plugin/kibana/app/kibana#/home?_g=())

AWS: Login via your ASW Okta tile

DVS Data Explorer: <https://dvs-production.vnerd.com/dvs-data-explorer>

Product Catalog: Login via Okta tile

# Technology KPI Tracking

Tracking spreadsheet:  [Dark Matter - Software Team Metrics](#)

Support-escalated issues tracking spreadsheet:  [Support-Escalated Issues](#)

Deployment frequency/change failure rate:

<https://devops.vnerd.com/Deployments/Teams/Report/Dark Matter>

Coding days/commit frequency:

[https://flow.pluralsight.com/v2/org/pluralsight/r/code-fundamentals-bc/coding\\_days/119957](https://flow.pluralsight.com/v2/org/pluralsight/r/code-fundamentals-bc/coding_days/119957)

PR time to merge:

<https://flow.pluralsight.com/v2/org/pluralsight/r/pr-resolution?team=119957>

## Learning

 [F# for fun and profit](#)

## Plan Management (Dark Matter) Front End Coding Standards

## Files/Organization

- Do name files using kebab-case
- Do add context to file names for searchability

Example

## TypeScript/JavaScript

- Do not use default exports, use named exports instead
- Do use TypeScript instead of vanilla JavaScript whenever possible
- Do prefer function over const declarations for components and longer functions

Example

- When importing types, do use `import type`

Example

## React

- Do create small reusable components that do one thing
- Do use `@pluralsight/mui` instead of homemade components when possible
- Avoid class components

## Data fetching

- Do put data fetching code in `lib/api`
- Do name data fetching files with `.api` such as `teams.api.ts`
- Prefer re-fetching data instead of caching across pages except for plan and user data

- Do use zod to parse returned API data

Example

## Presentation components

- Do not make decisions in presentation components
- Do take in individual props instead of complex components
- Avoid presentation components with many props, except top-level components (components right underneath a container)
- Do use component state for simple state that does not need to be shared above a component

## Utility components

- Do use utility components to do data-agnostic rendering such as layout
- Avoid passing too many props through the utility component to the components being laid out, instead, take in components themselves as props
- Avoid utility components with many props

## Redux

- Do group actions, sagas, and reducers together into files based on units of work
- Do name files grouping actions, sagas, and reducers with `.effects`, such as `edit-member-email.effects.ts`
- Do use sagas for non-data-fetching side effects
- Avoid using the Redux store directly

- Do reference `@reduxjs/toolkit` instead of directly importing from `redux`

## Connecting components to Redux

- Do prefer `useSelector` and `useDispatch` for connecting new components to `redux`
- Do use `connect` when memoizing and limiting renders is important for performance optimizations (when necessary)
- Do create types for store state and dispatch props via inference when using `connect`

### Example

- Avoid calculations in components, defer to Redux instead
- Do not connect to data that the component does not directly use (whenever possible)
  - Do not connect to data only to pass it down to children
- Do not use decorators to connect components

## Selectors

- Do test nontrivial selectors
- Do derive state via selector composition whenever possible
- Do keep selectors pure (no side effects)
- Do use `createSelector` from `@reduxjs/toolkit`
- Do create generic/configurable higher order selectors
- Do prefix selector names with `select`
- Do group multiple selectors into a single file
- Do name selector files with `.selectors`, such as `member.selectors.ts`

## Actions

- Do suffix the payload type with `Payload`
- Do keep types defined for action creators in the same file as the action creator
- Do not do complex logic in action creators
- Do use `namespace/camelCaseActionName` for action types, and use a common prefix for related actions

Example

- Do define actions using `createAction` from `@reduxjs/toolkit`

Example

- Do Use `createAction` from `@reduxjs/toolkit` for creating actions, and always reference the action creator itself rather than raw strings for action types

Example

- Don't depend on hard-coded action type values, prefer referencing action creators when possible

Example

- Do not suffix action creators with `Action`

Example

- Do name action creators as verbs

Example

## Thunks

- Do use `createAsyncThunk` for data fetching

Example

- Do put thunks in `*.actions.ts` files
- Do use the generated pending, fulfilled, and/or rejected actions to handle thunk resolution

Example

- Do not directly dispatch error actions in thunks, since this will be handled automatically

Example

## Sagas

- Do use sagas to respond to actions when a reducer is insufficient
- Do test sagas using `redux-saga-test-plan`
- Do not do mapping logic in sagas, defer to selectors instead
- Do suffix saga names with `Saga`
- Do create watcher sagas that register actions with sagas next to saga implementations
- Do prefix watcher saga names with `watch`
- Do dispatch analytics events from sagas
- Do compose saga exports via all function using named object syntax

Example

## Reducers

- Prefer inline handlers using `createReducer` from `@reduxjs/toolkit`
- Prefer putting core logic in reducers rather than in effects
- Do test complex reducers
- Do compose simple reducers instead of creating single large reducers

- Prefer using draft state modification over spreading state

Example

- Do use builder notation rather than object-style for reducers

Example

## Testing

- Do use msw to mock server data to test api calls in side effect code

Example

- Do assert using expect-style assertions

Example

## Styling

- Do use `sx` prop for simple one-off style customizations and in helper components
- Prefer using `createUseStyles` for styling components when classes can be reused or overridden
- Do use `styled` from `@mui/material` to create reusable variants of built-in elements or `@pluralsight/mui` components when the only changes are styles
- Avoid global styles whenever possible
- Do use `@pluralsight/mui` theme values instead of hard-coded values whenever possible
- Do talk to the team's UX designer if provided designs conflict with `@pluralsight/mui`



# Plan Management (Dark Matter) F# Coding Standards

- Do name functions as verbs
- Avoid redundancy in function names

Example

- Prefer to name types, modules, interfaces, and classes as nouns

Example

- Do not combine namespace and module declarations

Example

- Avoid unnecessary parenthesis

Example

- Avoid `|>` operator for simple, individual, single-line, single-argument functions

Example

- Prefer `|>` operator for passing data through multiple functions

Example

- Do decorate discriminated unions with `RequireQualifiedAccess` attribute

Example

- Do inject function dependencies before data dependencies

Example

- Do utilize currying when possible and reasonable

Example

## File naming

- Do add context to file names for discoverability

Example

# Business events

- Do put business events in their own project named after their main publishing Manager

Example

- Do name business events in past tense and suffix with Event

Example

- Do not use Manager operation DTOs on business events

Example

# Service contracts

- Do put service contracts in their own project separate from the service implementation

Example

- Do separate facets into their own subfolder/sub-namespace

Example

- Do separate DTOs into their own subfolder/sub-namespace

Example

- Avoid redundancy in DTO names already captured by facet

Example

- Do return Tasks from operations

Example

- Do create aliases for operation Results

Example

- Do create “constructor” helper functions for criteria DTOs where most values will not be set

Example

- Do not put logic in “constructor” helper functions
- Do create function aliases for interface operations to allow for easier consumption of the service

Example

# Service implementations

- Do put service implementations in their own project separate from the business events and contract

Example

- Do separate facets into their own subfolder/sub-namespace

Example

- Do separate pure logic functions into their own file/module

Example

- Do separate mapping into its own function in a dedicated file

Example

- Do use partial namespaces to reference DTOs in lower layers

Example

- Do use service implementation classes as composition roots for functions

# Managers

- Do not propagate external (manager) DTOs into engines, access, or utilities
- Do not leak internal (engine, access, or utility) DTOs out of managers
- Do not call other managers directly from a manager. Enqueue business events instead
- Do authorize manager operations

Example

- Do verify the existence of any resource IDs passed into a manager before passing them to engines and access

Example

# Engines

- Do not call managers from engines
- Do not call other engines from engines
- Do not perform side effects other than querying data from engines

# Access

- Do not leak resource implementation details from access
- Do not call managers from access
- Do not call engines from access
- Do not call other access from access

# Utilities

- Do not put business logic in utilities

# Testing

- Prefer integration testing managers over clients
- Structure tests in a way to limit the need to modify the tests if the signature of the function-under-test changes
- Use descriptive sentences for test names

Example

- Do use `randVal()` to generate random values for testing

Example

- Avoid [`<AutoData>`] and [`<AutoDataFSharp>`] when possible

Example

- Do use [`<InlineData>`] if known values are needed to be utilized to test values where the rest of the test remains the same

## Plan Management (Dark Matter) Misc. Team Engineering Guidelines Git strategy

We prefer a rebase branching strategy. So if you're working on branch feature and need to incorporate the latest changes from the main branch, once you've pulled down all changes you can run `git rebase main` or `git rebase origin/main` if you haven't pulled the latest changes to your local main branch yet.

You can also set your `git config pull.rebase=true` to ensure that if there are any merge conflicts, pulling will prompt a rebase instead of a merge commit. This helps

preserve the order of code changes and should help you avoid confusing merge issues and large merge commits.

When merging PRs, we prefer to squash the commits to keep a linear git history isolated to PRs whenever possible.

## PR Naming Conventions

We don't have any strict rules about commit or PR messages (we have talked briefly in the past about using semantic commit messages, but never really did anything about it), but by convention historically we write commit messages and PR titles in the present tense, e.g. instead of `Added color options to Button`, it would be `Add color options to Button`. All PRs must also link to a Jira ticket to conform with company-wide OKRs about change management.

## Leaving Code Better than you Found it

Often when going in to add a feature or fix a bug, you may encounter Tech Debt or code that could/should otherwise be refactored. Where appropriate, if it is necessary to make changes to a file to accomplish a task, it may make sense to pay off some tech debt at the same time since you should have a relatively high amount of context for that code at that time. However, we still want PRs to be as small in size and scope as possible, so if you identify Tech Debt that can be cleaned up related to your current work, if it is more than a trivial change, we prefer to split out that work into a separate PR so that we can isolate the core purpose of each change i.e. one PR for the Bug Fix, another PR that refactors related code to make it more readable. It is important that we don't just let our code bloat and get stale, so in addition to intentional larger Tech Debt tasks, we value

addressing Tech Debt as it comes up, where possible and where it does not get in the way of core tasks or project plans.

## Dark Matter DVS V1 to V2 Conversion

# Prerequisites

1. Setup DVS to run locally
  - a. Clone [dp-composer](#)
  - b. Change directory to `dp-composer`
  - c. Update the port for the postgres service in `docker-compose.yml` to run on any port but 5432
  - d. postgres
  - e. :
  - f. ...
  - g. ports
  - h. :
  - i. -
  - j. 5433
  - k. :
  - l. 5432
  - m. ...
  - n. Execute `docker-compose up -d`
  - o. Execute `docker-compose ps` and ensure all services are up and healthy
2. Run `Plans.WebEvents`
  - a. Change directory to `plan-management/docker`
  - b. Execute `docker-compose ps` to ensure all services are up and healthy. Ensure `rabbit-queue-manager` is in a running (healthy) state. If it is not, see 2.b.i
  - c. `$ docker-compose ps`
  - d. NAME COMMAND SERVICE STATUS PORTS
  - e. `docker-api-mocker-1 "npm start" api-mocker running 0.0.0.0:3030->3030/tcp`

- f. `docker-identity-lite-1 "docker-entrypoint.s..."`  
`identity-lite running 0.0.0.0:3100->3000/tcp`
- g. `docker-postgres-1 "docker-entrypoint.s..." postgres`  
`running 0.0.0.0:5432->5432/tcp`
- h. `docker-rabbit-1 "docker-entrypoint.s..." rabbit running`  
`4369/tcp, 5671/tcp, 0.0.0.0:5672->5672/tcp, 15671/tcp,`  
`25672/tcp, 0.0.0.0:15672->15672/tcp`
- i. `docker-rabbit-queue-manager-1 "./docker-entrypoint..."`  
`rabbit-queue-manager running (healthy)`  
`0.0.0.0:15000->3000/tcp`
  - i. Execute `docker-compose down` && `docker-compose up` to reset the Docker containers
  - ii. Verify the services are now healthy
  - iii. Run `Plans.CreateDatabase` to seed the database
- j. Run `Plans.WebEvents`. Wait until you see the message Service started.... If you encounter any errors, seek help from a teammate
- k. Navigate to <http://localhost:8088/v2/schemas>. You should see a bunch of `exp.plans.*` schemas returned

# Converting the topic

1. Find the V1 schema you are going to convert in the code. The schema will live in `PlanManagement.Manager.Notification.Service` and will have `Dvs.V1.Topic` attribute. For example:
2. `[<Dvs.V1.Topic("Team",`  
`Dvs.DataClassification.InternalUseOnly, Version = 3)>]`
3. `type TeamDvsMessageV3 =`
4. `{`
5. `[<Dvs.Key(Index = 1)>]`
6. `Id: string`
7. `[<Dvs.Key(Index = 0)>]`
8. `PlanId: string`
9. `ParentTeamId: string option`



- 10.Name: string
- 11.Description: string option
- 12.EventName: string
- 13.EventDate: DateTimeOffset
- 14.}
- 15.Take a moment to read through the schema and understand all of the fields
- 16.Find usages of the V1 schema and understand which scenarios publish to the topic
- 17.Create the V2 version of the schema
  - a. Create two new types: one for the key and one for the value
  - b. Copy and adjust the fields to match the new schema
    - i. Ensure any dates are updated to be in the new XyzAt format
  - c. Implement the Dvs.V2.IValue<> interface on the value type
  - d. Add Dvs.Default attributes on any optional fields
  - e. Add Dvs.Documentation attributes to all fields
    - i. As much as possible, re-use documentation from other V2 schemas to ensure consistency across V2 topics
    - ii. Be sure to call out any interesting nuances of the data
    - iii. For any field that can be joined with another topic, call out the join in the documentation
    - iv. Cross-reference the documentation with the V1 documentation, located at  
<https://github.com/ps-dev/dont-panic/wiki/{V1TopicName}>
    - v. Your topic should look similar to:
    - vi. type DvsTeamV1Key =
    - vii. {
    - viii.[<Dvs.Documentation("The plan the team is associated with. " + Documentation.planJoin)>]
    - ix. PlanId: string
    - x. [<Dvs.Documentation "The unique identifier for the team. Team IDs are reused during plan merges.">]
    - xi. Id: Guid
    - xii. }
    - xiii.[<Dvs.V2.Topic("Team", 1u,  
Dvs.DataClassification.InternalUseOnly)>]

```

xiv.[<Dvs.Documentation "Represents a team in an
    organizational hierarchy.">]
xv.type DvsTeamV1Value =
xvi.{
xvii. [<Dvs.Default null>]
xviii. [<Dvs.Documentation "Optional parent team that
    the team is nested under.">]
xix.ParentTeamId: Guid option
xx.[<Dvs.Documentation "Customer-defined name of the
    team. Team names will be unique across a plan.">]
xxi.Name: string
xxii. [<Dvs.Default null>]
xxiii. [<Dvs.Documentation "Optional customer-defined
    description.">]
xxiv. Description: string option
xxv. [<Dvs.Documentation "The timestamp when the
    team was created.">]
xxvi. CreatedAt: DateTimeOffset
xxvii. [<Dvs.Documentation "The timestamp when the
    team was last updated.">]
xxviii. UpdatedAt: DateTimeOffset
xxix. }
xxx. interface Dvs.V2.IValue<DvsTeamV1Key> with
xxxi. member this.CreatedAt = this.CreatedAt
xxxii. member this.UpdatedAt = this.UpdatedAt
xxxiii. type DvsTeamV1 = Dvs.V2.Payload<DvsTeamV1Key,
    DvsTeamV1Value>

```

18. Update all usages of the V1 schema to also publish the new V2 schema. Use the dependencies.DvsQueue.EnqueueV2 function
19. Run the tests in PlanManagement.Manager.Notification.Test.Unit. Update tests as needed to account for the new V2 publishing
20. Add a Dvs.Documentation attribute to the V1 topic indicating that the V2 topic replaces it. For example:
21. [<Dvs.V1.Topic("Team",  
Dvs.DataClassification.InternalUseOnly, Version = 3)>]

22. [<Dvs.Documentation "Deprecated. Replaced by  
skills.plans.v1.Team">]
23. type TeamDvsMessageV3 =
24. ...
25. Run Plans.WebEvents
26. Navigate to <http://localhost:8088/v2/topics>. Verify your V2 topic shows up

# Testing the topic locally

1. Create a local replication job for the V2 topic:

```
POST localhost:8080/v2/jobs/
```

2. {
3. "jobName"
4. :
5. "SomeJobName"
6. ,
7. "startingOffsets"
8. :
9. "earliest"
10. ,
11. "consumerGroupName"
12. :
13. "SomeJobName"
14. ,
15. "notificationUrl"
16. :
17. "myNotifUrl.com"
18. ,
19. "source"
20. :
21. {
22. "topicName"
23. :

```
24. "skills.plans.v1.Team"
25. }
26. ,
27. "teamName"
28. :
29. "Dark Matter"
30. ,
31. "sink"
32. :
33. {
34. "sinkType"
35. :
36. "postgres"
37. ,
38. "tableName"
39. :
40. "skills_plans_v1_team"
41. ,
42. "host"
43. :
44. "postgres"
45. ,
46. "port"
47. :
48. 5432
49. ,
50. "database"
51. :
52. "postgres"
53. ,
54. "user"
55. :
56. "postgres"
57. ,
58. "password"
```

- 59. :
- 60. "postgres"
- 61. ,
- 62. "ssl"
- 63. :
- 64. false
- 65. }
- 66. }
- 67. Run the app locally and initiate all use cases that trigger data to be published to the V2 topic
- 68. Verify that records for the V2 topic are getting written to the replicated PostgreSQL table
- 69. Once you are satisfied that the V2 publishing is working correctly, create a pull request with your code changes

## Writing backfill script(s)

1. Create an ad-hoc that backfills the current state of the database into the V2 topic
  - a. Query all plans, and then for each plan, query the data for the topic in order to minimize the chance of stale data being backfilled into the topic
  - b. For each piece of data representing a record in the topic, call the DVS publisher for that data
  - c. Reference the [skills.plans.v1.Team backfill ad-hoc](#) for an example of a current state backfill script
2. If the topic is a historical topic, you must create a second ad-hoc for backfilling the historical data
  - a. You will end up with two ad-hocs, for example:  
BackfillSkillsPlansV1LearnerHistory\_CurrentState and  
BackfillSkillsPlansV1LearnerHistory\_HistoricalState
  - b. Query removed records from a replication table for the V1 version of the topic
  - c. Create a dynamic, user-specified cutoff date. This will change based on the environment and will be specified when the ad-hoc is run. A simple way to do this is to make the date a configuration setting the App.config file for AdHocCommands

- d. Publish each removed record older than the cutoff date to the V2 topic
3. Create a pull request with your ad-hoc backfill script(s)

# Publishing and backfilling in staging

1. Ensure your pull requests have been approved, merged, built in CI/CD, and deployed to staging
2. Verify data is being published to the V2 topic
  - a. Go to  
<https://dvs-staging.vnerd.com/dvs-data-explorer/topics?selected={V2TopicName}>
  - b. Click on the *Records* tab on the right side of the screen and verify there are records being written
  - c. If there is not any data yet, you can use the UI to trigger some data to be published
3. If your topic is historical, find the earliest removal date published to the topic
  - a. Scan the data in the *Records* tab for the earliest removal date. There should only be a few records since this is a new topic
  - b. Record the earliest removal date. You will need it later
4. Run the current-state backfill ad-hoc
  - a. Ensure the `commandToRun` in `Program.cs` of `AdHocCommands` is set to your current-state backfill ad-hoc
  - b. Build `AdHocCommands` in *Release* mode
  - c. Create a Zip file of the `plan-management/Plans/AdHocCommands/bin` directory and name the file the same as the backfill ad-hoc
  - d. Create a Remote Desktop Connection to the staging ad-hoc server machine
    - i. The host is  
`isw-plan-management-adhoc-1.eplur-staging.vnerd.com`
    - ii. The username is Administrator
    - iii. The password can be found in the *Plan Management - Nodes* note in LastPass, in the line directly under the host

- e. Copy the Zip file to the D:\adhocs directory on the server
  - f. Extract the Zip file
  - g. Run AdHocCommands.exe inside the extracted directory
  - h. Ensure the command specified after Running command in the console is the backfill ad-hoc
  - i. Press enter to start the ad-hoc
  - j. If the ad-hoc throws any exceptions, reach out to a teammate for help
5. Verify the current-state data was backfilled correctly
- a. Create a replication job for the V2 topic
    - i. Set the jobName and consumerGroupName to be  
plan-management-self-replication---{V2TopicName}---v2
    - ii. Set source.topicName to be the V2 topic name
    - iii. Set sink.tableName to be  
dvs2\_\_sr\_\_skills\_\_plans\_\_{V2EntityName}\_\_v1
    - iv. Set sink.host and sink.password based on the values from the  
*Plan Management - RDS* note in LastPass  
POST <https://hydra-streams-staging.vnerd.com/v2/jobs>
    - v. {
    - vi. "jobName"
    - vii. :
    - viii. ,
    - ix. "consumerGroupName"
    - x. :
    - xi. ,
    - xii. "teamName"
    - xiii.:
    - xiv."Dark Matter"
    - xv. ,
    - xvi."startingOffsets"
    - xvii. :
    - xviii. "earliest"
    - xix. ,
    - xx. "notificationsUrl"
    - xxi.:
    - xxii. "hydra-notifications-stage.vnerd.com:8080/notif  
y/slack?channel=plan-management-replication"
    - xxiii. ,

xxiv. "source"  
xxv. :  
xxvi. {  
xxvii. "topicName"  
xxviii. :  
xxix. }  
xxx. ,  
xxxi. "sink"  
xxxii. :  
xxxiii. {  
xxxiv. "sinkType"  
xxxv. :  
xxxvi. "postgres"  
xxxvii. ,  
xxxviii. "tableName"  
xxxix. :  
xl. ,  
xli. "host"  
xlii. :  
xlili. ,  
xliv. "port"  
xlv. :  
xlvi. 5432  
xlvii. ,  
xlviii. "user"  
xlix. :  
l. "dvs\_replication"  
li. ,  
lii. "password"  
liii. :  
liv. ,  
lv. "database"  
lvi. :  
lvii. "plan\_management"  
lviii. ,



- lix. "schema"
- lx. :
- lxi. "replication"
- lxii. ,
- lxiii. "ssl"
- lxiv. :
- lxv. false
- lxvi. }
- lxvii. }

b. Verify the data backfilled into the V2 topic matches the data in the source of truth database

- i. Find the latest creation date in the database. Round the date up to the nearest minute to avoid precision issues
- ii. Count all records in the source of truth database with a creation date earlier than the date above
- iii. Count all records in the replicated V2 topic with a creation date earlier than the date above
- iv. Verify the two counts match. Reach out to a teammate for help if they do not
- v. Example SQL queries:
- vi. select
- vii. creation\_date
- viii. from
- ix. ps
- x. .
- xi. teams
- xii. where
- xiii. removal\_date
- xiv. is
- xv. null
- xvi. order
- xvii. by
- xviii. creation\_date
- xix. desc
- xx. limit
- xxi. 1

```

xxii. ;
xxiii. -- Returns 2022-11-07 23:21:48.669782 +00:00
xxiv. select
xxv. count
xxvi. (
xxvii. *
xxviii. )
xxix. from
xxx. ps
xxxi. .
xxxii. teams
xxxiii. where
xxxiv. removal_date
xxxv. is
xxxvi. null
xxxvii. and
xxxviii. creation_date
xxxix. <
xl. '2022-11-07 23:21:49.000000 +00:00'
xli. select
xlii. count
xliii. (
xliv. *
xlv. )
xlvi. from
xlvii. replication
xlviii. .
xlix. dvs2__sr__skills__plans__team__v1
l. where
li. created_at
lii. <
liii. '2022-11-07 23:21:49.000000 +00:00'

```

# Publishing and backfilling in production

1. Ensure the backfill(s) for the V2 topic have been run in staging
2. Ensure the code changes for the V2 topic have been deployed to production
3. Verify data is being published to the V2 topic
  - a. Go to  
<https://dvs-production.vnerd.com/dvs-data-explorer/topics?selected={V2TopicName}>
  - b. Click on the *Records* tab on the right side of the screen and verify there are records being written
  - c. If there is not any data yet, keep refreshing every few minutes until there is data
4. If your topic is historical, find the earliest removal date published to the topic
  - a. Scan the data in the *Records* tab for the earliest removal date. There should only be a few records since this is a new topic
  - b. Record the earliest removal date. You will need it later
5. Run the current-state backfill ad-hoc
  - a. Ensure the `commandToRun` in `Program.cs` of `AdHocCommands` is set to your current-state backfill ad-hoc
  - b. Build `AdHocCommands` in *Release* mode
  - c. Create a Zip file of the `plan-management/Plans/AdHocCommands/bin` directory and name the file the same as the backfill ad-hoc
  - d. Create a Remote Desktop Connection to the staging ad-hoc server machine
    - i. The host is `ipw-plan-management-adhoc-1.vnerd.com`
    - ii. The username is `Administrator`
    - iii. The password can be found in the *Plan Management - Nodes* note in LastPass, in the line directly under the host
  - e. Copy the Zip file to the `D:\adhocs` directory on the server
  - f. Extract the Zip file
  - g. Run `AdHocCommands.exe` inside the extracted directory
  - h. Ensure the command specified after `Running` command in the console is the backfill ad-hoc

- i. Press enter to start the ad-hoc
  - j. If the ad-hoc throws any exceptions, reach out to a teammate for help
- 6. Verify the current-state data was backfilled correctly
  - a. Create a replication job for the V2 topic
    - i. Set the jobName and consumerGroupName to be  
plan-management-self-replication---{V2TopicName}---v2
    - ii. Set source.topicName to be the V2 topic name
    - iii. Set sink.tableName to be  
dvs2\_\_sr\_\_skills\_\_plans\_\_{V2EntityName}\_\_v1
    - iv. Set sink.host and sink.password based on the values from the  
*Plan Management - RDS* note in LastPass  
POST <https://hydra-streams-production.vnerd.com/v2/jobs>
    - v. {
    - vi. "jobName"
    - vii. :
    - viii. ,
    - ix. "consumerGroupName"
    - x. :
    - xi. ,
    - xii. "teamName"
    - xiii. :
    - xiv. "Dark Matter"
    - xv. ,
    - xvi. "startingOffsets"
    - xvii. :
    - xviii. "earliest"
    - xix. ,
    - xx. "notificationsUrl"
    - xxi. :
    - xxii. "hydra-notifications.vnerd.com:8080/notify/slack?channel=plan-management-replication"
    - xxiii. ,
    - xxiv. "source"
    - xxv. :
    - xxvi. {
    - xxvii. "topicName"

xxviii. :  
xxix. }  
xxx. ,  
xxxi. "sink"  
xxxii. :  
xxxiii. {  
xxxiv. "sinkType"  
xxxv. :  
xxxvi. "postgres"  
xxxvii. ,  
xxxviii. "tableName"  
xxxix. :  
xl. ,  
xli. "host"  
xlii. :  
xlili. ,  
xliv. "port"  
xlv. :  
xlvi. 5432  
xlvii. ,  
xlviii. "user"  
xlix. :  
l. "dvs\_replication"  
li. ,  
lii. "password"  
liii. :  
liv. ,  
lv. "database"  
lvi. :  
lvii. "plan\_management"  
lviii. ,  
lix. "schema"  
lx. :  
lxi. "replication"  
lxii. ,

lxiii. "ssl"

lxiv. :

lxv.false

lxvi. }

lxvii. }

b. Verify the data backfilled into the V2 topic matches the data in the source of truth database

i. Find the latest creation date in the database. Round the date up to the nearest minute to avoid precision issues

ii. Count all records in the source of truth database with a creation date earlier than the date above

iii. Count all records in the replicated V2 topic with a creation date earlier than the date above

iv. Verify the two counts match. Reach out to a teammate for help if they do not

v. Example SQL queries:

vi. select

vii. creation\_date

viii. from

ix. ps

x. .

xi. teams

xii. where

xiii. removal\_date

xiv. is

xv. null

xvi. order

xvii. by

xviii. creation\_date

xix. desc

xx. limit

xxi. 1

xxii. ;

xxiii. -- Returns 2022-11-07 23:21:48.669782 +00:00

xxiv. select

xxv. count

```

xxvi. (
xxvii. *
xxviii. )
xxix. from
xxx. ps
xxxi. .
xxxii. teams
xxxiii. where
xxxiv. removal_date
xxxv. is
xxxvi. null
xxxvii. and
xxxviii. creation_date
xxxix. <
xl. '2022-11-07 23:21:49.000000 +00:00'
xli. select
xlii. count
xlili. (
xliv. *
xlv. )
xlvi. from
xlvii. replication
xlviii. .
xlix. dvs2__sr__skills__plans__team__v1
l. where
li. created_at
lii. <
liii. '2022-11-07 23:21:49.000000 +00:00'

```

Add label

## DVS Validation Scripts

### **skills.plans.v1.Leader**

```

select count(*) from ps.plan_users where removal_date is null
and leader_id is not null;

```

```

select count(*) from
replication.dvs2__sr__skills__plans__leader__v1;
select *
from ps.plan_users pu
join replication.dvs2__sr__skills__plans__leader__v1 r on r.id =
pu.leader_id
where r.plan_user_id::text <> pu.id or
r.created_at::timestamp(0) <>
pu.leader_assignment_date::timestamp(0);
select count(*)
from replication.dvs2__sr__skills__plans__leader__v1 r
where managed_team_ids <> '{}' and manager_permission_set_id is
not null;
select count(*)
from ps.plan_users pu
where teams @> '[{"isManaged": true}]' and leader_id is not null
and removal_date is null;
select *
from replication.dvs2__sr__skills__plans__leader__v1 r
join ps.plan_users pu on pu.leader_id = r.id and pu.teams @>
'[{"isManaged": true}]'
where not r.managed_team_ids @> (select
array_agg(team->>'teamId')::uuid[]
from ps.plan_users as pu2, jsonb_array_elements(pu2.teams) as
team
where (team->>'isManaged')::bool is true and pu2.id = pu.id) or
(not r.managed_team_ids <@ (select
array_agg(team->>'teamId')::uuid[]
from ps.plan_users as pu2, jsonb_array_elements(pu2.teams) as
team
where (team->>'isManaged')::bool is true and pu2.id = pu.id));
select *
from replication.dvs2__sr__skills__plans__leader__v1 r
join ps.plan_users pu on pu.leader_id = r.id and pu.teams @>
'[{"isManaged": true}]'

```



```
where r.manager_permission_set_id <> (select id from
ps.permissions p where p.id = any(pu.leader_permission_ids) and
p.level = 'managed-teams');
```

### **skills.plans.v1.Learner**

VerifySkillsPlansV1Learner

AdHoc from branch verification-adhocs

+

```
select count(*) from
replication.dvs2__sr__skills__plans__learner__v1;
select count(*) from ps.plan_users where removal_date is null
and learner_id is not null;
select *
from replication.dvs2__sr__skills__plans__learner__v1 r
join ps.plan_users pu on r.id = pu.learner_id
where pu.id <> r.plan_user_id::text or
pu.learner_assignment_date::timestamp(0) <>
r.created_at::timestamp(0);
select *
from replication.dvs2__sr__skills__plans__learner__v1 r
left join ps.plan_users pu on r.id = pu.learner_id and
pu.removal_date is null
where pu.id is null
```

### **skills.plans.v1.LearnerHistory**

```
select count(*) from
replication.dvs2__sr__skills__plans__learner_history__v1 where
removed_at is null;
select count(*) from ps.plan_users where removal_date is null
and learner_id is not null and not is_pending;
select *
from replication.dvs2__sr__skills__plans__learner_history__v1 r
```

```

join ps.plan_users pu on r.id = pu.learner_id and
pu.removal_date is null
where r.user_handle <> pu.redeemed_identity_id or r.plan_id <>
pu.plan_id or r.created_at::timestamp(0) <>
pu.learner_assignment_date::timestamp(0) or r.removed_at is not
null;

```

#### **skills.plans.v1.LearnerInviteHistory**

```

select count(*) from
replication.dvs2__sr__skills__plans__learner_invite_history__v1
where canceled_at is null and redeemed_at is null;
select count(*) from ps.plan_users where removal_date is null
and learner_id is not null and is_pending;
select *
from
replication.dvs2__sr__skills__plans__learner_invite_history__v1
r
join ps.plan_users pu on r.id = pu.learner_id and
pu.removal_date is null
where r.expires_at::timestamp(0) <>
pu.pending_expiration_date::timestamp(0) or
r.is_canceled or
r.redeemed_by is not null or
r.sent_at::timestamp(0) <> pu.pending_send_date::timestamp(0) or
r.created_at::timestamp(0) <>
pu.learner_assignment_date::timestamp(0) or
r.plan_id <> pu.plan_id or
r.sent_to <> pu.pending_email;

```

#### **skills.plans.v1.PermissionSet**

```

select count(*)
from ps.permissions pe
join plans p on pe.plan_id = p.id
where not p.is_archived;
select count(*)
from replication.dvs2__sr__skills__plans__permission_set__v1;

```

```

with permissions as (select p.id,
p.plan_id,
p.name,
p.level,
p.rule_ids,
p.creation_date,
d.permission_id is not null as is_default
from ps.permissions p
left join ps.default_permissions d on p.plan_id = d.plan_id and
p.id = d.permission_id
)
select *
from replication.dvs2__sr__skills__plans__permission_set__v1 r
join permissions p on r.id = p.id
where p.is_default <> r.is_default or r.level <> p.level or
r.name <> p.name or r.plan_id <> p.plan_id or
r.created_at::timestamp(0) <> p.creation_date::timestamp(0);

```

### **skills.plans.v1.Plan**

```

select count(*) from
replication.dvs2__sr__skills__plans__plan__v1;
select count(*) from plans where not is_archived;
select *
from replication.dvs2__sr__skills__plans__plan__v1 r
join plans p on p.id = r.id
join subscriptions s on p.id = s.plan_id
where
r.display_name <> p.display_name or
r.created_at::timestamp(0) <> p.created_date::timestamp(0) or
r.is_pilot <> s.is_pilot or
r.expires_at::timestamp(0) <> s.expiration_date::timestamp(0) or
r.is_suspended <> (p.suspended_date is not null);
with features as (
select j.plan_id, array_agg(distinct j.feature_ids) as
feature_ids

```

```

from
(select s.plan_id, unnest(p.feature_ids) as feature_ids
from subscriptions s,
jsonb_array_elements(s.products) as e(product),
ps.products p
where p.id = e.product->>'id'
) j
group by j.plan_id
)
select *
from replication.dvs2__sr__skills__plans__plan__v1 r
join features s on s.plan_id = r.id
where r.feature_ids::text[] <> s.feature_ids;
with flags as (
select p.id, jsonb_build_object('user_skill_level_restricted',
p.is_user_skill_level_restricted,
'license_allocation_summary_restricted',
p.is_license_allocation_summary_disabled) as flags
from plans p
)
select *
from replication.dvs2__sr__skills__plans__plan__v1 r
join flags f on f.id = r.id
where r.flags <> f.flags;
with products as (
select j.plan_id, array_agg(distinct j.products) as products
from
(select s.plan_id, jsonb_build_object('product_id', p.id,
'product_option_id', e.product->>'optionId',
'total_license_count', case when p.is_assignable then
(e.product->>'purchasedQuantity')::integer else null::integer
end) as products
from subscriptions s,
jsonb_array_elements(s.products) as e(product),
ps.products p

```

```

where p.id = e.product->>'id'
) j
group by j.plan_id
)
select *
from replication.dvs2__sr__skills__plans__plan__v1 r
join products s on s.plan_id = r.id
where r.products <> s.products;

```

### **skills.plans.v1.PlanUser**

```

select count(*) from ps.plan_users where removal_date is null;
select count(*) from
replication.dvs2__sr__skills__plans__plan_user__v1;
select *
from replication.dvs2__sr__skills__plans__plan_user__v1 r
join ps.plan_users pu on pu.id = r.id::text
where r.is_pending <> pu.is_pending or r.user_handle <>
pu.redeemed_identity_id or r.created_at::timestamp(0) <>
pu.creation_date::timestamp(0) or r.plan_id <> pu.plan_id or
r.pending_email <> pu.pending_email or r.note <> pu.note;

```

### **skills.plans.v1.PlanUserTeamHierarchy**

VerifySkillsPlansV1PlanUserTeamHierarchy

AdHoc from branch verification-adhocs

+

```

select count(*) from ps.plan_users where removal_date is null
and teams @> ' [{"isManaged": false}]';
select count(*) from
replication.dvs2__sr__skills__plans__plan_user_team_hierarchy__v
1;
select *

```

```

from
replication.dvs2__sr__skills__plans__plan_user_team_hierarchy__v
1 r
join ps.plan_users pu on r.plan_user_id::text = pu.id
where r.created_at::timestamp(0) <>
pu.creation_date::timestamp(0) or not pu.teams @>
'["isManaged": false]';

```

### **skills.plans.v1.Team**

```

select count(*) from ps.teams where removal_date is null;
select count(*) from
replication.dvs2__sr__skills__plans__team__v1;
select *
from replication.dvs2__sr__skills__plans__team__v1 r
join ps.teams t on t.id = r.id::text and t.plan_id = r.plan_id
where r.parent_team_id::text <> t.parent_team_id or
r.description <> t.description or r.name <> t.name or
r.created_at::timestamp(0) <> t.creation_date::timestamp(0);
Add label

```

## Query 'User CSV Table Download' Manually

Just a head's up, you'll need to lookup the planId and leader permission Id's and replace them in this query!

```

WITH column_names AS (
SELECT
'PlanUserId' AS PlanUserId,
'Email' AS Email,
'Note' AS Note,
'Teams' AS Teams,
'IsAdmin' AS IsAdmin,
'IsManager' AS IsManager,
'ManagerPermission' AS ManagerPermission,
'TeamsManaged' AS TeamsManaged,

```

```

'BusinessEnterpriseLicense' AS BusinessEnterpriseLicense,
'LabsLicense' AS LabsLicense,
'TechFoundationsAddOnLicense' AS TechFoundationsLicense,
'Action' AS Action
),
data_rows AS (
SELECT
COALESCE(p.id::text, '') AS PlanUserId,
COALESCE(u.email, '') AS Email,
COALESCE(p.note, '') AS Note,
COALESCE((
SELECT STRING_AGG(t.name, ';')
FROM jsonb_array_elements(p.teams) AS e(team)
JOIN ps.teams t ON t.plan_id = 'capgemini-service-sas-824a6' AND
t.id = (e.team->>'teamId')
), '') AS Teams,
CASE WHEN '53a21ed3-7479-4f77-818a-8febb89c08cb'::uuid =
ANY(p.leader_permission_ids) THEN 'true' ELSE 'false' END AS
IsAdmin,
CASE WHEN p.leader_permission_ids &&
ARRAY['911a0c40-3a0d-4eea-8789-f64a0f20c133',
'87e80dd6-cf8d-4774-a8e0-5ca24614fa2d',
'c1d6395d-46db-4dba-bb2d-d08e44025fd2']::uuid[] THEN 'true' ELSE
'false' END AS IsManager,
CASE
WHEN '911a0c40-3a0d-4eea-8789-f64a0f20c133' =
ANY(p.leader_permission_ids) THEN 'Full Manager'
WHEN '87e80dd6-cf8d-4774-a8e0-5ca24614fa2d' =
ANY(p.leader_permission_ids) THEN 'Basic Manager'
WHEN 'c1d6395d-46db-4dba-bb2d-d08e44025fd2' =
ANY(p.leader_permission_ids) THEN 'Limited Manager'
ELSE ''
END AS ManagerPermission,
COALESCE((
SELECT STRING_AGG(t.name, ';')

```

```

FROM jsonb_array_elements(p.teams) AS e(team)
JOIN ps.teams t ON t.plan_id = 'capgemini-service-sas-824a6' AND
t.id = (e.team->>'teamId')::text AND
(e.team->>'isManaged')::bool is true
), '') AS TeamsManaged,
EXISTS (
SELECT 1
FROM ps.products product
WHERE product.id = ANY(p.learner_product_ids)
AND product.name = 'Business Enterprise'
)::text AS BusinessEnterpriseLicense,
EXISTS (
SELECT 1
FROM ps.products product
WHERE product.id = ANY(p.learner_product_ids)
AND product.name LIKE '%Labs%'
)::text AS LabsLicense,
EXISTS (
SELECT 1
FROM ps.products product
WHERE product.id = ANY(p.learner_product_ids)
AND product.name LIKE '%Tech - Foundations%'
)::text AS TechFoundationsAddOnLicense,
'' AS Action
FROM
ps.plan_users AS p
JOIN
replication.dvs2__tech__identity__user__v4 AS u ON
p.redeemed_identity_id = u.handle
WHERE
p.plan_id = 'capgemini-service-sas-824a6'
)
SELECT * FROM (
SELECT * FROM column_names
UNION ALL

```



```
SELECT * FROM data_rows  
) AS result;
```

## Sign In Via SSO

This will work on staging and prod environments.

I'll describe the staging process:

1. Go to the [staging login page](#).
2. Click "Sign in with company or school (SSO)."
3. Use **pluralsightoauth** as the company URL.
4. Sign in using your pluralsight Google account.

## SSO Bug Reconciliation

Aug 31, 2023 1 min read 4 people viewed

[Simple solution for the SSO login bug](#)  
[SSO Bug Summary](#)

## Simple solution for the SSO login bug

Data we need: Get most recent product assignment for all users who logged in via SSO

- before bug
- (active) users after bug began.

Comparisons we'll make:

- If product assignments are  $> 0$  after bug began, leave these users be. They have licenses.
- If product assignment = 0 after bug began and product assignment = 0 before bug, leave these users be. They shouldn't have licenses assigned to them.

- If product assignment = 0 after bug began and product assignment > 0 before bug, restore products to users. Products are found in the before bug query (and are most recent assignments, so can be confident that these are their rightful product assignments).
- If product assignment = 0 after bug began and a most recent product assignment cannot be found for this user in before bug, they are assumed to be a new user and should be issued the plan's default products.

## Process

### Getting users who logged in during the sso bug:

```
sso_logins_during_bug AS (

SELECT *

FROM "DVS"."HISTORY"."EXP_IDENTITY_USERSIGNIN"

WHERE "TIMESTAMP" BETWEEN '2023-08-21 16:00:00.0-06:00' AND
'2023-08-24 15:30:00.0-06:00'

AND "AUTHENTICATIONMETHOD" = 'Sso'

),

joined_data AS (

SELECT

pu.planid,

pu.userhandle,
```

```

i.email,

l.products,

l.id,

l.updatedat,

lh.removedat,

lh.createdat

FROM "DVS"."CURRENT_STATE"."SKILLS_PLANS_V1_PLANUSER" pu

JOIN "DVS"."CURRENT_STATE"."TECH_IDENTITY_V4_USER" i ON i.handle
= pu.userhandle

JOIN "DVS"."CURRENT_STATE"."SKILLS_PLANS_V1_LEARNERHISTORY" lh
ON lh.userhandle = pu.userhandle

LEFT JOIN sso_logins_during_bug s ON s.handle = pu.userhandle

LEFT JOIN "DVS"."HISTORY"."SKILLS_PLANS_V1_LEARNER" l ON
l.planuserid = pu.id

WHERE pu.id IS NOT NULL

AND s.handle IS NOT NULL

),

```

From the users we retrieved above,

**Let's get their most recent product assignments before the bug**

```
most_recent_update_before_bug as (  
  
SELECT  
  
planid,  
  
userhandle,  
  
email,  
  
products as productassignmentattimeofupdate,  
  
id,  
  
updatedat  
  
FROM (  
  
SELECT *,  
  
ROW_NUMBER() OVER (PARTITION BY userhandle ORDER BY updatedat  
DESC) AS rn  
  
FROM joined_data  
  
where updatedat < 1690159200000 -- unix epoch milliseconds of  
when the bug began  
  
) ranked_data  
  
WHERE rn = 1  
  
ORDER BY userhandle, updatedat DESC
```

),

Now,

**Let's get most recent product assignments that occurred after the bug began**

most\_recent\_update\_after\_bug\_began as (

SELECT

planid,

userhandle,

email,

products as productassignmentattimeofupdate,

id,

updatedat

FROM (

SELECT \*,

ROW\_NUMBER() OVER (PARTITION BY userhandle ORDER BY updatedat  
DESC) AS rn

FROM joined\_data

where updatedat >= 1690159200000

) ranked\_data

```
WHERE rn = 1
```

```
AND removedat is null -- making sure users haven't been removed  
since bug
```

```
ORDER BY userhandle, updatedat DESC
```

```
)
```

## SSO Bug Summary

**\*As of *9/7/2023 3pm EST*, wrongfully  
revoked licenses have been restored.**

## SSO Bug timeframe

Started ~2023-08-21 16:00:00.0-06:00 UTC

Fixed ~2023-08-24 15:30:00.0-06:00 UTC

## What happened?

Users who logged in via SSO during this time had their licenses removed.

## How we got this data

This data comes from a mishmash of DVS data, queried via Snowflake. Here's the gist:

```
WITH
```

```
sso_logins AS (
```

```

SELECT *
FROM "DVS"."HISTORY"."EXP_IDENTITY_USERSIGNIN"
WHERE "TIMESTAMP" BETWEEN '2023-08-21 04:00:00.0-06:00' AND
'2023-08-24 03:30:00.0-06:00'
AND "AUTHENTICATIONMETHOD" = 'Sso'
),
prior_known_sso_logins AS (
SELECT *
FROM "DVS"."HISTORY"."EXP_IDENTITY_USERSIGNIN"
WHERE "TIMESTAMP" < '2023-08-21 04:00:00.0-06:00'
AND "AUTHENTICATIONMETHOD" = 'Sso'
),
usersWhoGotLicensesRemoved AS (
SELECT
pu.planid,
pu.userhandle,
i.email
FROM "DVS"."HISTORY"."SKILLS_PLANS_V1_PLANUSER" pu
JOIN "DVS"."CURRENT_STATE"."TECH_IDENTITY_V4_USER" i ON i.handle
= pu.userhandle
JOIN "DVS"."CURRENT_STATE"."SKILLS_PLANS_V1_LEARNERHISTORY" lh
ON lh.userhandle = pu.userhandle
JOIN sso_logins s ON s.handle = pu.userhandle

```

```

-- Left JOIN prior_known_sso_logins p ON p.handle = s.handle
--add to get new OR previous users. Remove if getting total
LEFT JOIN "DVS"."HISTORY"."SKILLS_PLANS_V1_LEARNER" l ON pu.id =
l.planuserid
LEFT JOIN "DVS"."CURRENT_STATE"."SKILLS_PLANS_V1_LEADER" leader
ON pu.id = leader.planuserid
--where s.handle is null --add to get new users. Remove if
getting total
--where p.handle is not null and s.handle is not null --add to
get previous users. Remove if getting total
WHERE ((l.id IS NULL AND isadmin IS NULL AND
managerPermissionSetId IS NULL) OR l.id IS NOT NULL)
AND lh.removedat is null
AND (l.id IS NULL
OR (ARRAY_SIZE("PRODUCTS") = 0 OR "PRODUCTS" IS NULL))
AND i.email NOT LIKE '%@pluralsight.com'
GROUP BY pu.planid, pu.userhandle, i.email
),
usersWhoAreLikelyNotStillImpacted AS (
SELECT
ur.planid,
ur.userhandle,
ur.email,
l.products

```



```

FROM usersWhoGotLicensesRemoved ur
JOIN "DVS"."CURRENT_STATE"."SKILLS_PLANS_V1_LEARNER" l ON
ur.userhandle = l.userhandle
WHERE ARRAY_SIZE(l.products) > 0
)
--SELECT DISTINCT u.* --uncomment to find users still impacted
--FROM usersWhoGotLicensesRemoved u
--LEFT JOIN usersWhoAreLikelyNotStillImpacted a ON u.userhandle
= a.userhandle
--WHERE a.userhandle IS NULL;
select distinct
r.planid,
r.userhandle,
r.email
from usersWhoGotLicensesRemoved r
order by r.planid, r.email

```

## What are we going to do to fix this?

### Option 1: The simple, less engineer-y way

Work with CSM's to manually add licenses back to users via using the UI and/or CSV uploads.

Pros:

- This seems like the fastest way to accomplish this. Because we're only dealing with ~**1,632** users who are in an incorrect state, we could probably tackle this as a team in a single workday.
- Because we're not rushing and human eyes would be on every user adjustment, creating any further errors would be avoidable.

Cons:

- This may be a mindless, tedious, annoying day of work.

## **Option 2: The automated, engineer-y way**

Step 1: Write a script to add products to impacted unlicensed users in our database based on current plan license defaults.

Step 2: Write a script to re-publish fixed user license data to DVS topics.

Pros:

- Honestly, we can't think of too many pros at this point. If this bug had impacted more users, an automated fix would make the most sense, but since the damage was relatively small, we don't believe the engineering resources needed for this type of fix is worthwhile to Pluralsight.

Cons:

- This could take 1-2 weeks to accomplish. There would be several steps we'd have to take such as creating scripts to revert the process in case something goes awry, writing the actual scripts, etc.

- A greater margin of error could exist. The only way that this would work is if we're 100 percent confident in our queried data, which it's difficult to know if there were edge cases.
- We're dealing with some really important accounts here and we don't want to do any further damage. When you run an automated script, you hope for the best, but there's always a chance that it could make things worse if there's any amount of miscalculation.

Add label

## On-Call Playbook

# Error Monitoring

See the Link pack for links

[Plan Management \(Dark Matter\) Link Pack | Monitoring](#)

- Check [#dark-matter-alerts](#)
- Ack OpsGenie alerts
- Investigate in NewRelic if applicable
  - Usually related to usage/usability issues
  - Often these resolve themselves, but not always, so don't just ignore them
- Look into Kibana logs to investigate root cause of bugs
- Check Rabbit queues
  - Shovel dead letter queues
  - This resolves many rabbit issues
  - If they come back into the dead letter, investigate further

# If a major issue is identified with a recent publish

If the Octopus deployment can be safely rolled back, and a fix can't be completed within a short window, it should be rolled back, along with the commit unless a fix will be merged the same day.

## Other Work

- Security Vulnerabilities
  - Check Snyk, resolve any outstanding issues