NEOMATIC

build your business with NEO experience

NEO.CPO Installation Guide

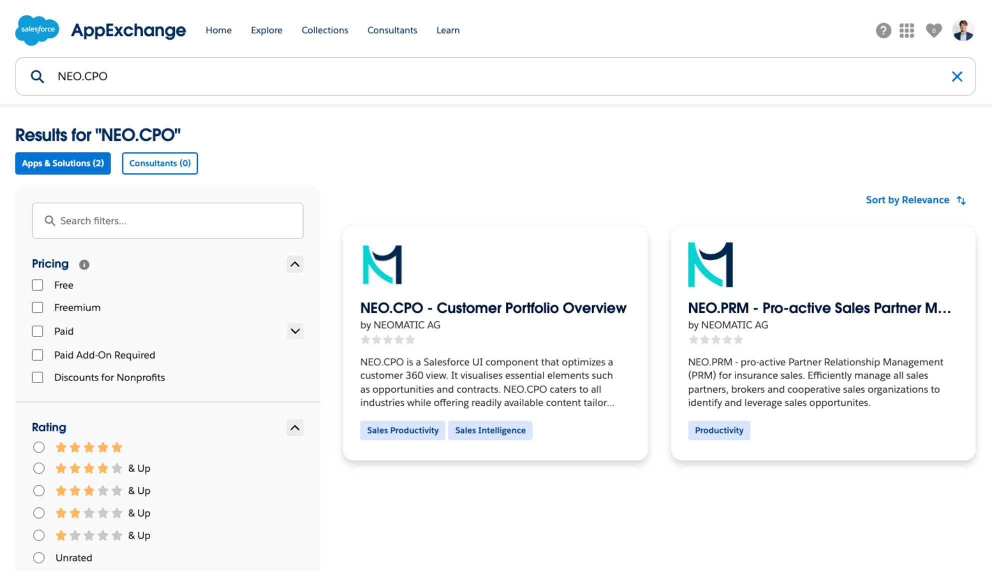
Table of Contents

Installation

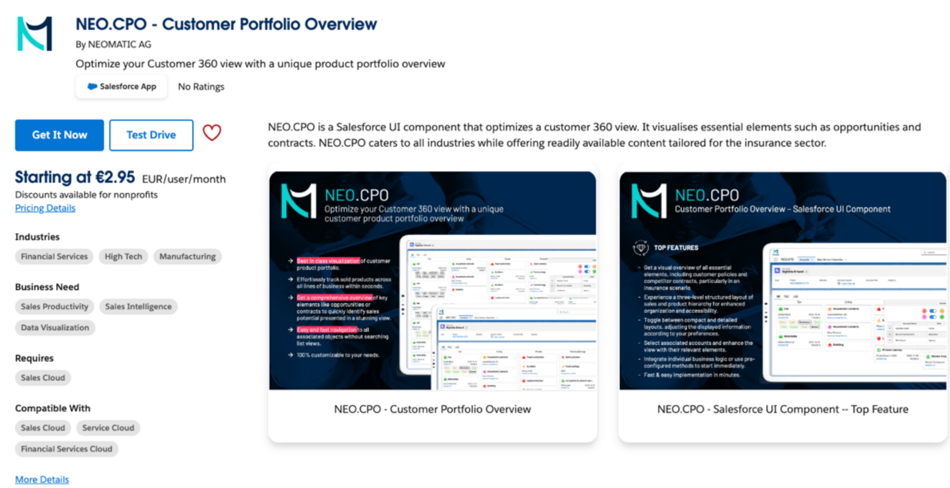
We only allow to install our Apps via the AppExchange.

Navigate to the AppExchange: https://appexchange.salesforce.com/

Search for NEO.CPO

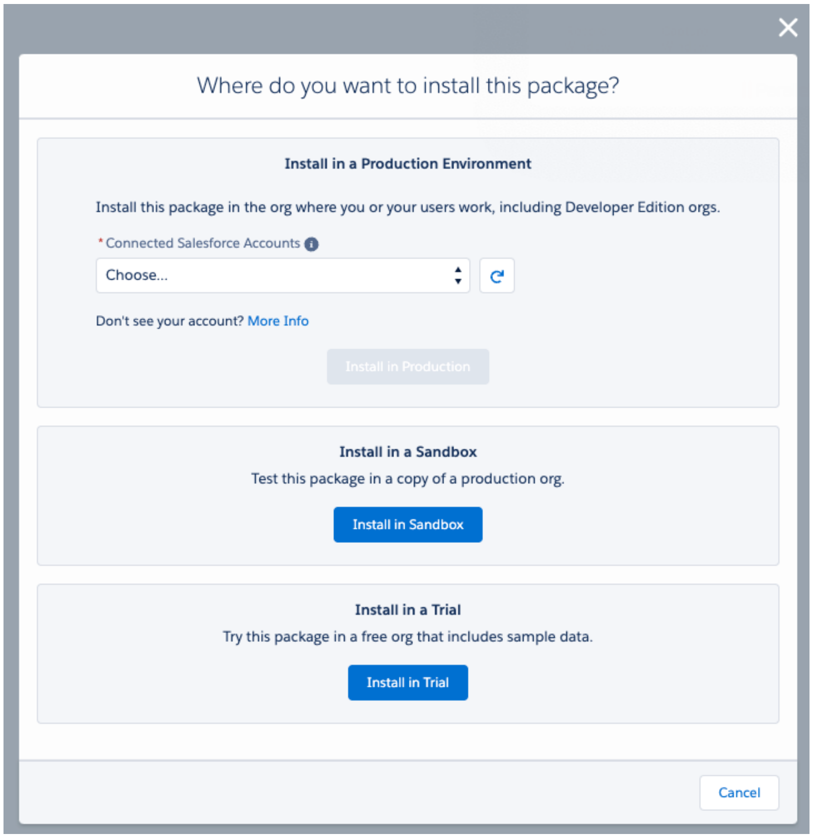


Click the “NEO.CPO - Customer Portfolio Overview” Tile.

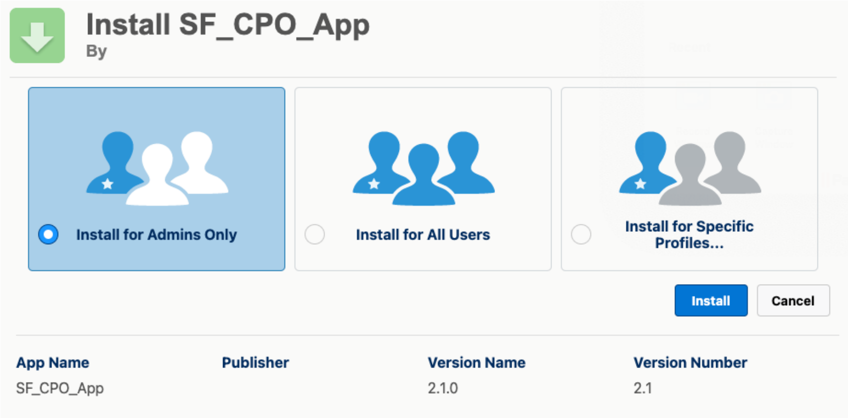


Click the “Get It Now” Button.

We recommend installing the App on your sandbox first.

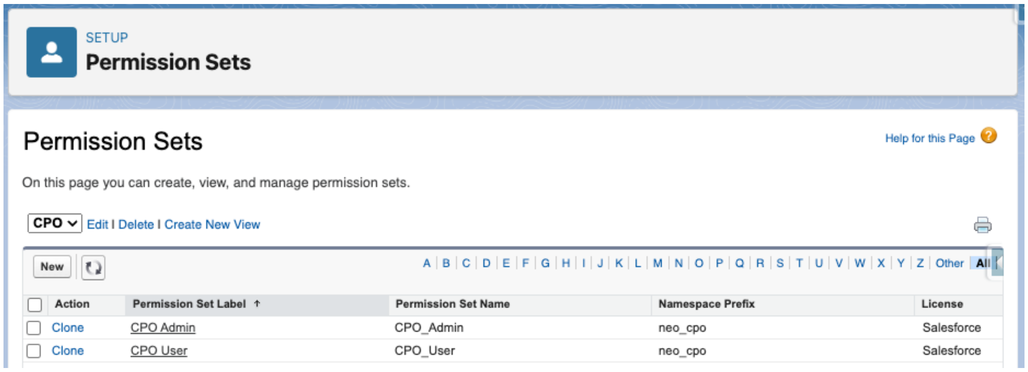


We recommend installing the App for Admins only.



Permission Sets

With our App you will receive two permission sets to be used to assign to your users.



The CPO Admin Permission Set is supposed to be assigned to users that would maintain the sales and service hierarchy. (Note: More about the sales and service hierarchy will be explained later)

The CPO User Permission Set is supposed to be assigned to your standard users, so that they can use the App.

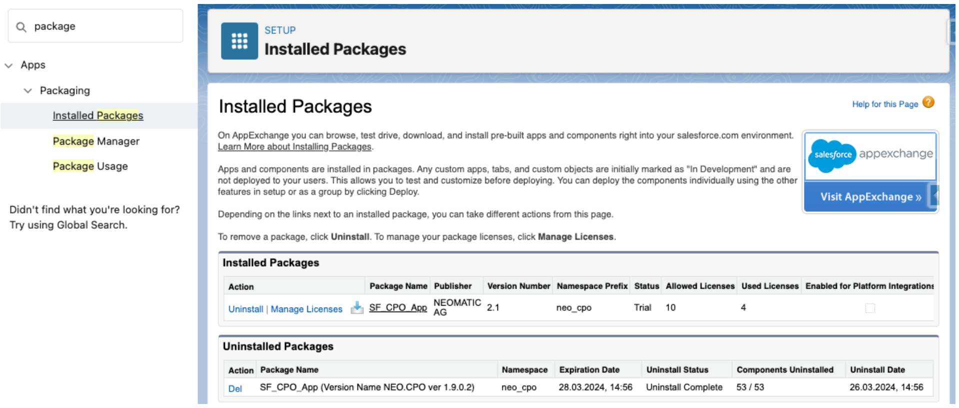
A user that is assigned to the system administrator profile does have already full access to the app per installation option you have chosen.

Licenses

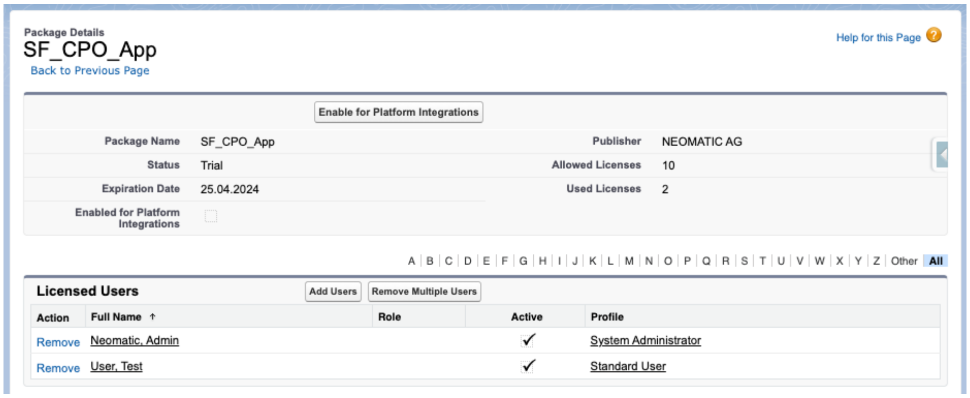
There is no license assignment needed on sandboxes. Additional steps may be necessary for productive orgs, depending on the licensing model you have chosen with us.

User-based license

For a user-based license model you need to manage licenses per user.



Click on Manage Licenses



Click on Add Users to add as many users as you have available licenses.

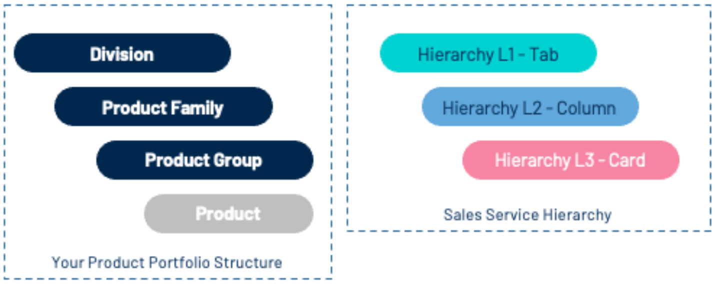
Org-wide license

If you have with org-wide license, you do not need to care about single user assignment as every active user of the org is entitled to use the App.

Maintaining the Sales Service Hierarchy

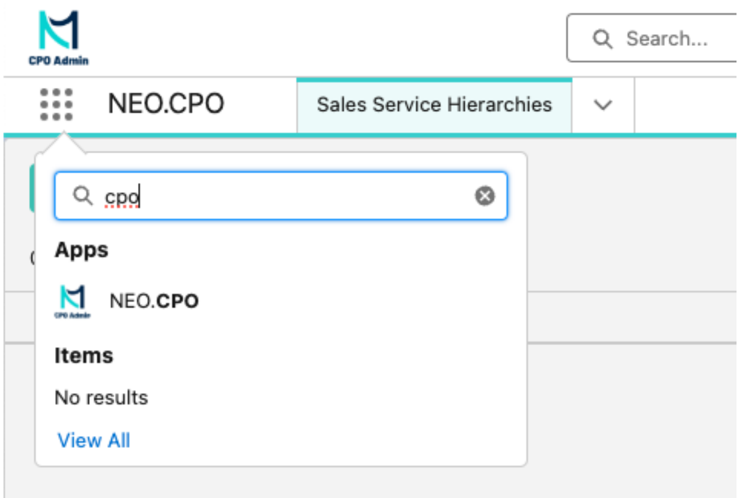
The Sales Service Hierarchy is mandatory for the App, hence need to be maintained. We recommend to create the hierarchy based on your product portfolio structure.

The following illustration shows you an example of a product portfolio structure combined with the corresponding hierarchy level setup.



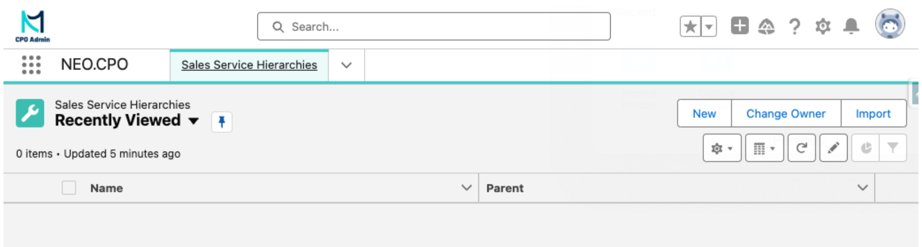
A full App configuration requires records for all hierarchy levels.

To maintain the sales service hierarchy you can use the NEO.CPO App to be opened from the App Launcher.

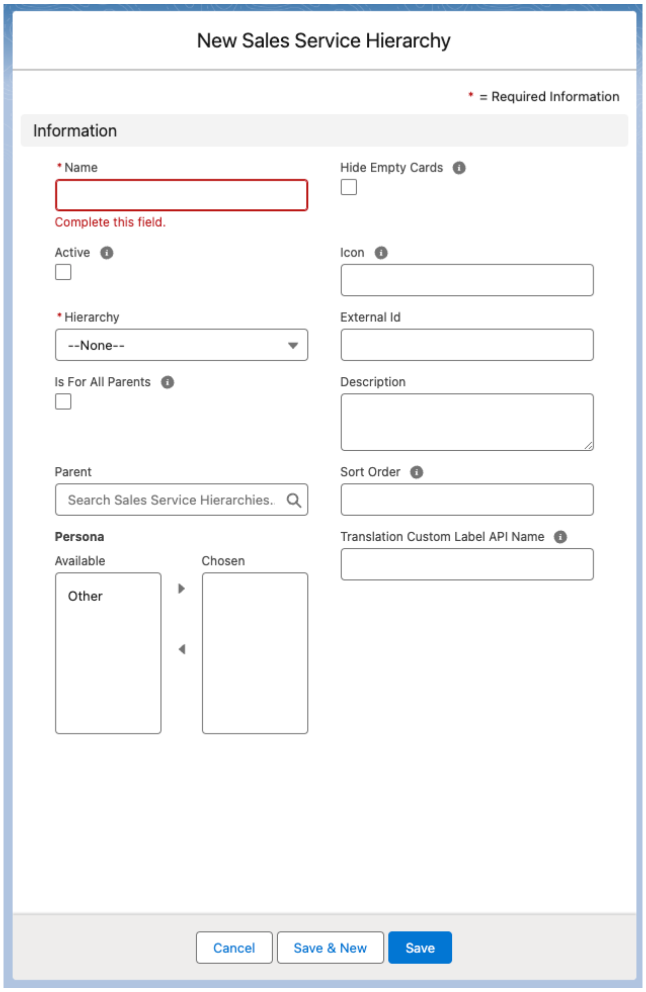


The initial tab of that console tab is for the Sales Service Hierarchy Object. As you can see there are no objects, yet.

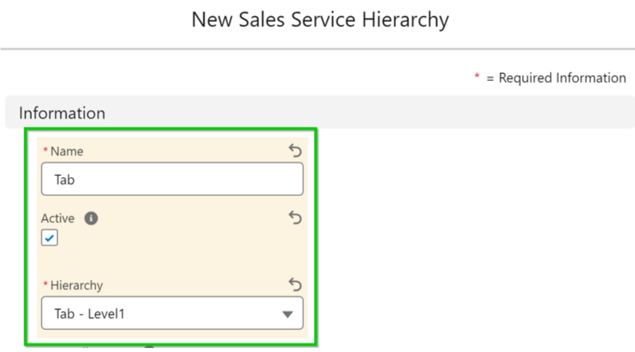
Record Creation



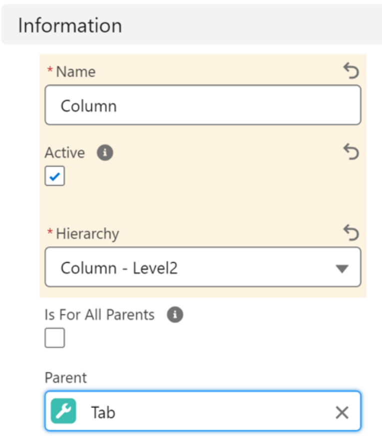
Click on New to create the first object.



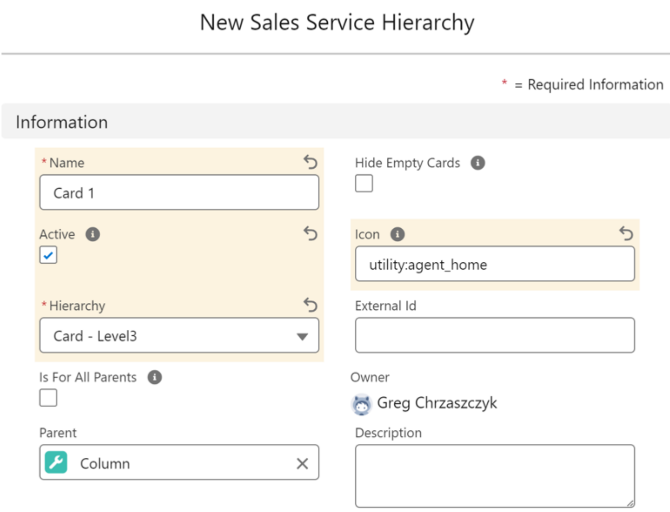
To define the first level fill SSH Name Field and select from picklist Hierarchy - **Tab - Level 1**. This level requires only three fields to be populated.



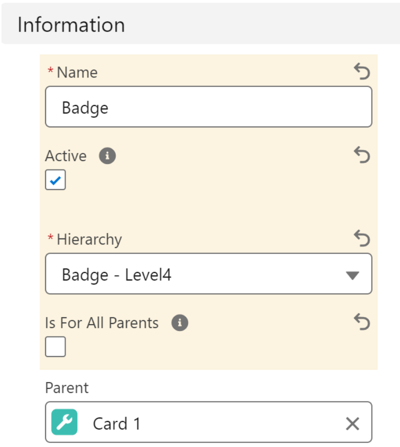
To define the second level, click on the previously created record → Related Tab → New Button and fill the SSH field Hierarchy - **Column - Level 2**. This level requires the same fields like level 2 but with additional Parent field which is prefilled if created from related list.



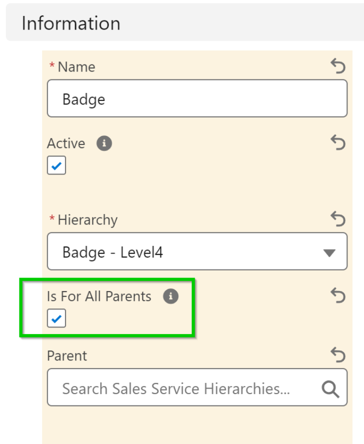
For the third level repeat the previous step with filling Hierarchy - **Card - Level 3** and additional to that add icon to each third level by inserting their name text from [Icons - Lightning Design System](). It is recommended to use the utility icons, for example: utility:agent_home



For the fourth level (optional) repeat the previous step with filling Hierarchy - **Badge - Level 4** but do not add Icons. You can either add Parent lookup to apply this Badge configuration on specific Card Level 3 or select **Is For All Parents** to apply it on all Level 3 cards.



or



To change the order of the tabs, columns, cards and badges within their respective place, fill an integer number into the field “Sort Order”, the lower the number, the further left or up a tab/column/card/tab will appear. It is recommended to use below range:

Tab - Level 1	Column - Level 2	Card - Level 3	Badge - Level 4
1-9	10-99	100-999	> 1000

And start numbering with:

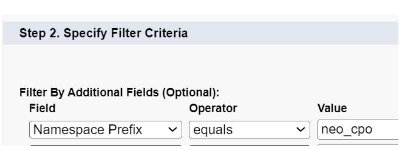
Tab - Level 1	Column - Level 2	Card - Level 3	Badge - Level 4
3	30	300	3000

Translations

NEO CPO App supports translations via standard Salesforce Custom Labels. By default English values are provided.

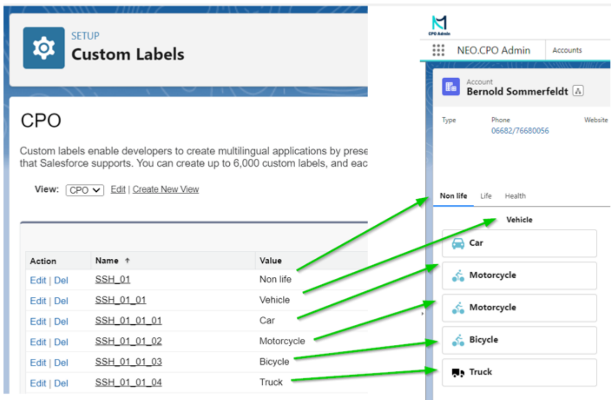
Use Translation Workbench to maintain translated values. Specify languages for translation and assign translators for each language. Manage translated values for any Salesforce supported language. Translators can maintain translations directly through the workbench, or you can export translation files for bulk translation imports.

You can create dedicated List View for CPO labels only with the filter:

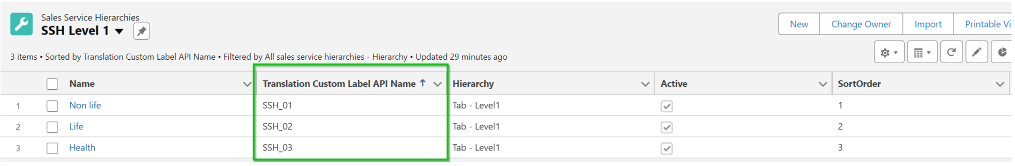


If you would like to translate your sales service hierarchy records, then please follow these steps.

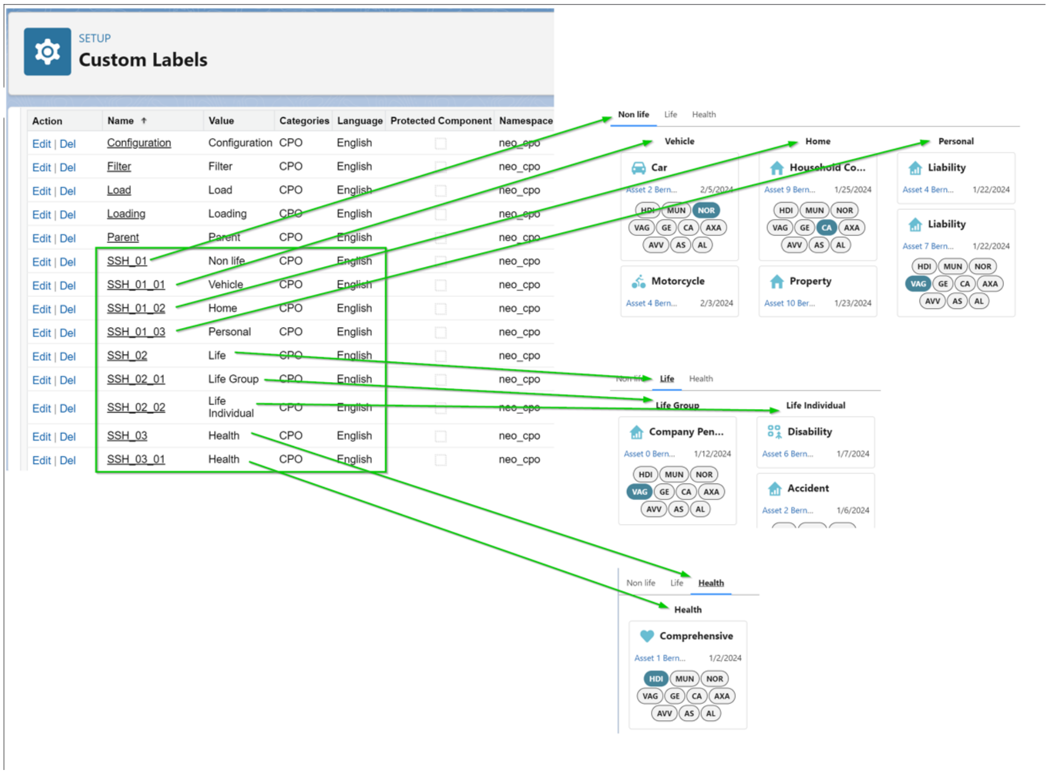
1. Create Custom Labels
2. Add Custom Label API name to your sales service hierarchy records



The easiest way to populate API names to your records is to use an editable list view on the object.

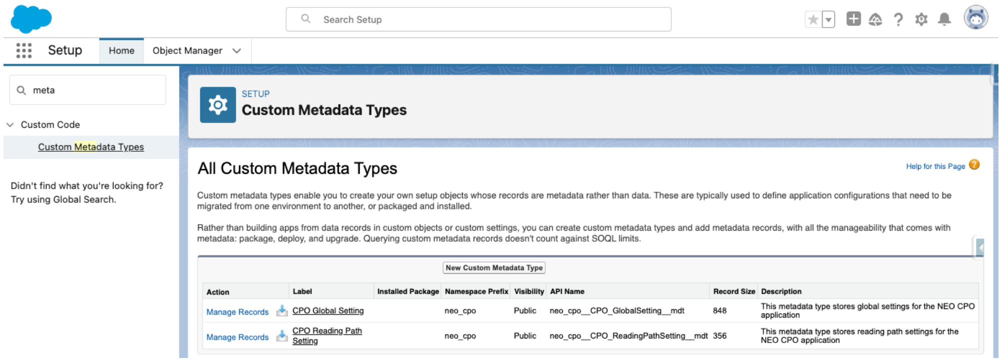


Example:

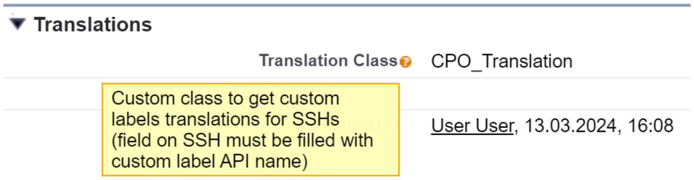


If you need some specific translation logic, you can add custom apex class for that.

The class name must be configured in the custom metadata settings for our App. Navigate to custom metadata types.



To add custom class Click on Manage Records for CPO Global Settings.



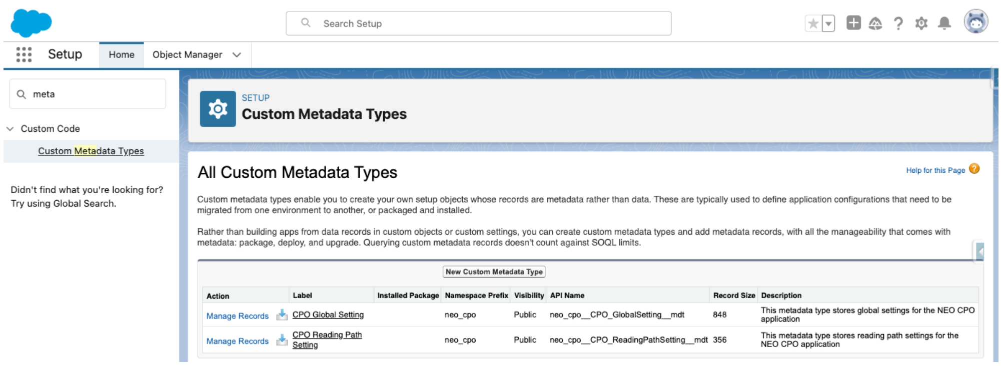
In the section translations add the class name of your custom translation class.

New Apex Class with this name must be deployed. Use below skeleton of custom translation class:

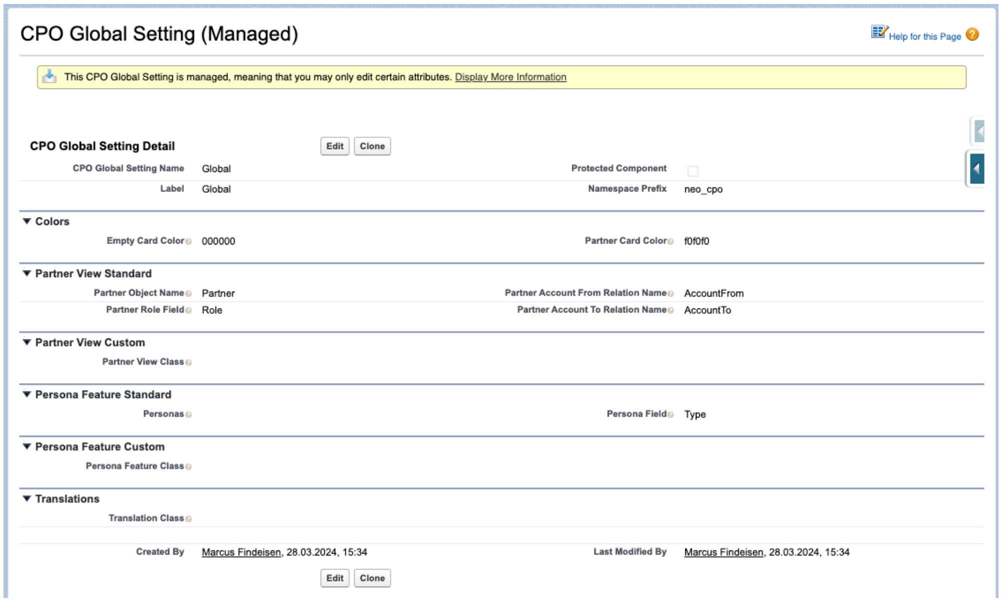
```
global with sharing class CPO_Translation implements Callable {
      global Object call(String param, Map<String, Object> args) {
            return System.Label
                  .get(
                        null,
                        args.keySet().iterator().next(),
                        UserInfo.getLanguage()
                  );
      }
}
```

Global App Settings

Navigate to custom metadata types.



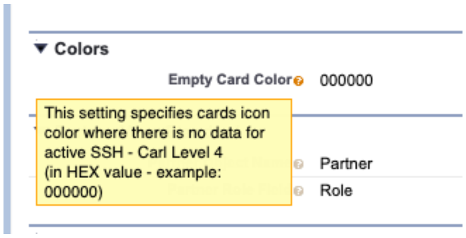
Click on Manage Records for CPO Global Settings.



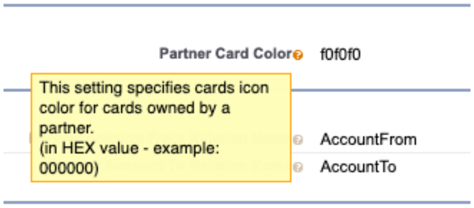
! DO NOT MODIFY **"Global"** record API name.
Only "**Global"** value is valid for CPO Global Setting Name!

Colouring

The App works in a way that every card where no match to any record does exists, the card shows up as an empty card. It is possible to define a colour as a hex-value, so that users can easily differ those cards from others.

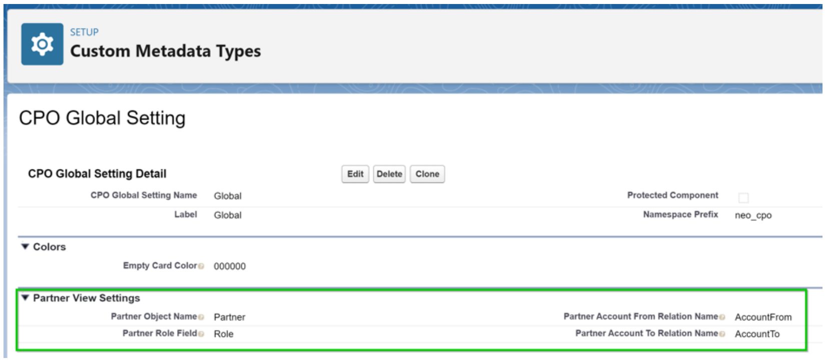


If the partner feature is activated, the App does also show cards from other root Objects such as other Accounts on an Account Record Page. With the other colour setting you can define a background highlight for those to differentiate them.



Partner View Feature

Default standard configuration considers standard Salesforce **Partner** object ([sforce_api_objects_partner]) with standard relation names.



Partner View feature is not active if Partner Object Name field on CPO Global Settings is empty. Remaining field values in Partner View Settings section on layout will not be considered at all in this case.

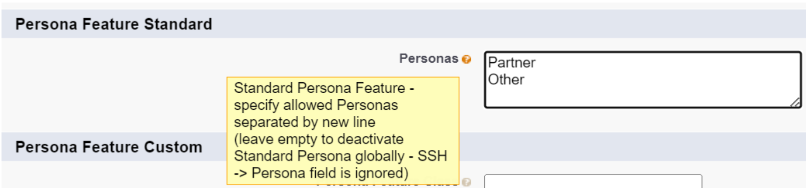
To deactivate Partner View feature, simply remove value from Partner Object Name field on CPO Global Settings and keep or remove remaining field values in Partner View Settings section.

Standard Partner View feature can be customized with other Standard or Custom Object to be used.

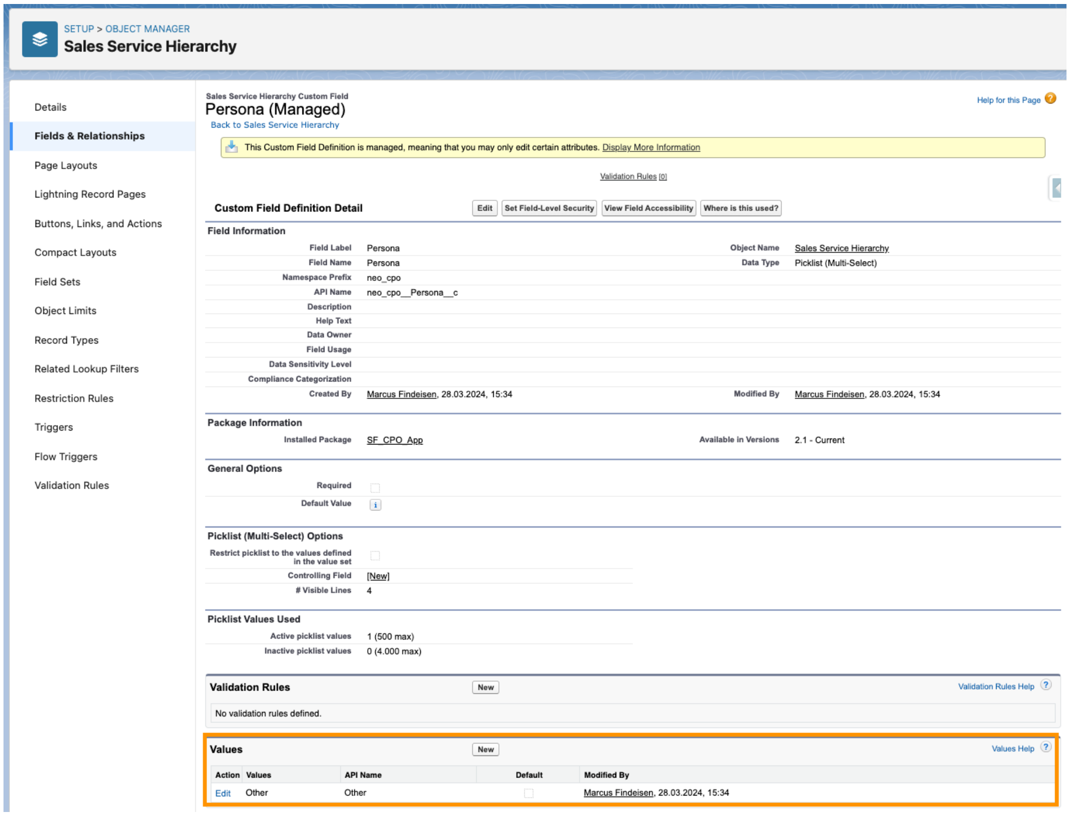
Persona Feature

This feature allows you to steer different CPO Dashboards for different account groupings. As part of our standard feature they are grouped by Account Type. You can customize that to any other field on Account.

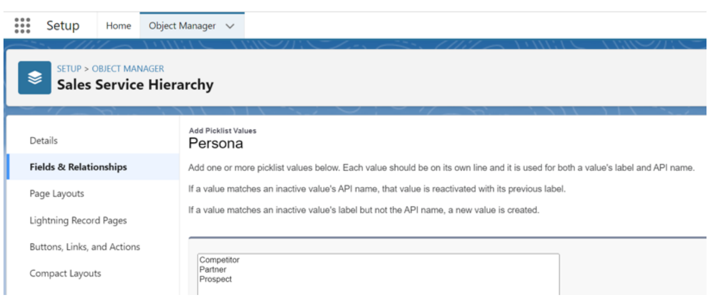
To use the feature you need to add the values of the selected Account field (in our standard Account Type) to the Persona field.



Besides of this you need to add those values to the Sales Service Hierarchy Object – Field Persona.

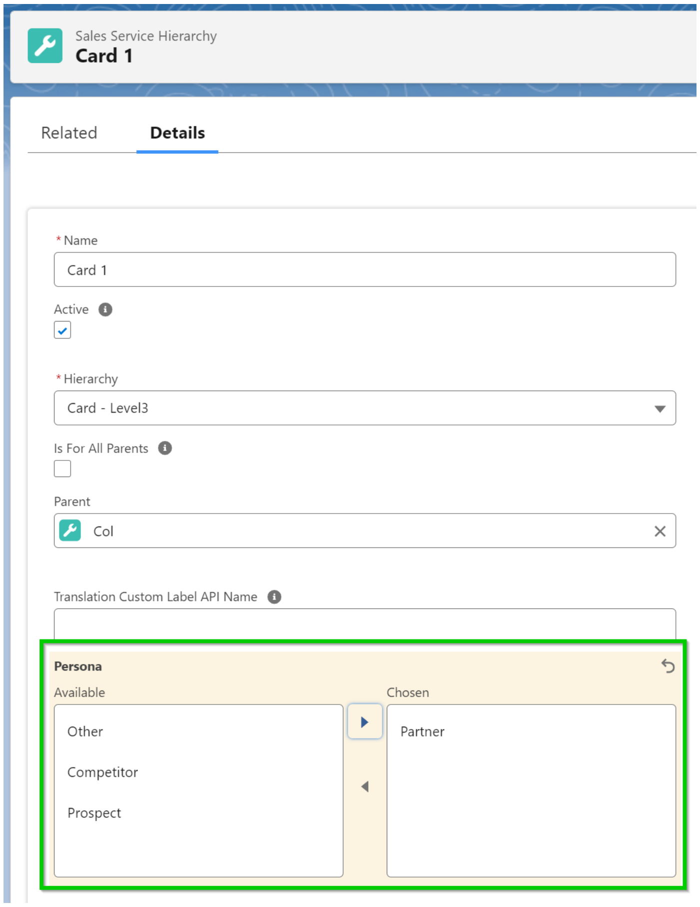


Click New



After you did these two steps you need to populate the persona value to your Sales Service Hierarchy Records. With that you can steer different views.

Open Card - Level 3 Sales Service Hierarchy record and add needed Personas to the possible picklist values of the field.

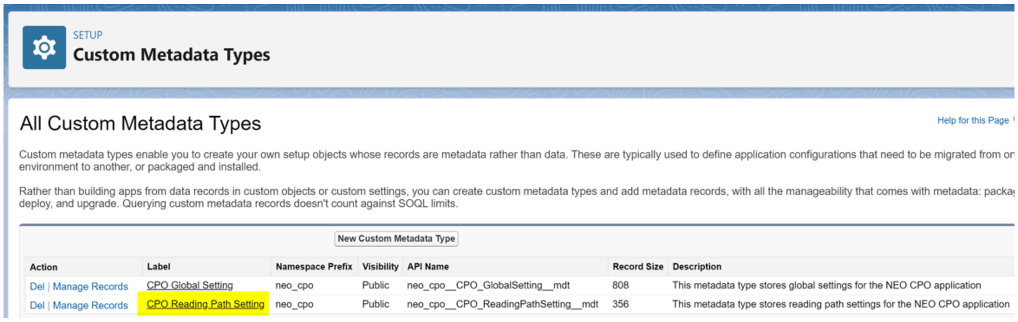


Reading Path Settings

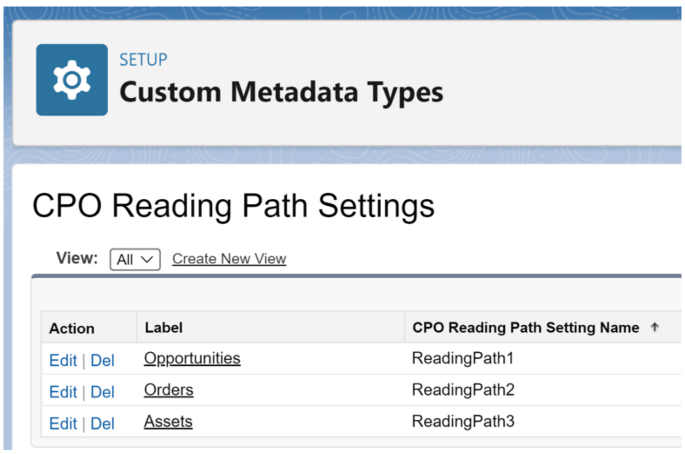
NEO CPO App provides out-of-the box three standard Reading Paths for standard objects:

- Opportunity
- Order
- Asset

All are configured via Custom Metadata Types: **CPO Reading Path Setting**



Click on Manage Records



Reading Path 1 - Opportunity

Data flow: Opportunity → Opportunity Line Item → Product → Sales Service Hierarchy

Reading Path 2 - Order

Data flow: Order → Order Item → Product → Sales Service Hierarchy

Reading Path 3 - Asset

Data flow: Asset→ Product → Sales Service Hierarchy

with **Warranty Assets object, field Warranty Type** used to render Badges on cards.

Note: Each reading path require that Sales Service Hierarchy records exist and lookup on Products -> Sales Service Hierarchy field is assigned

Standard Reading Path Considerations:

Access to all needed fields need to be granted for the user to render data specified in standard reading paths.

Standard reading path 1 **Opportunity fields**	**Standard reading path 2** **Order fields**	**Standard reading path 3** **Asset fields**
`AccountId`	`AccountId`	`AccountId`
`Amount`	`Name`	`Name`
`Description`	`OrderNumber`	`InstallDate`
`Name`	`ContractId`	`UsageEndDate`
`NextStep`	`TotalAmount`	`Price`
`StageName`	`EffectiveDate`	`PurchaseDate`
`Type`	`Description`	`Description`

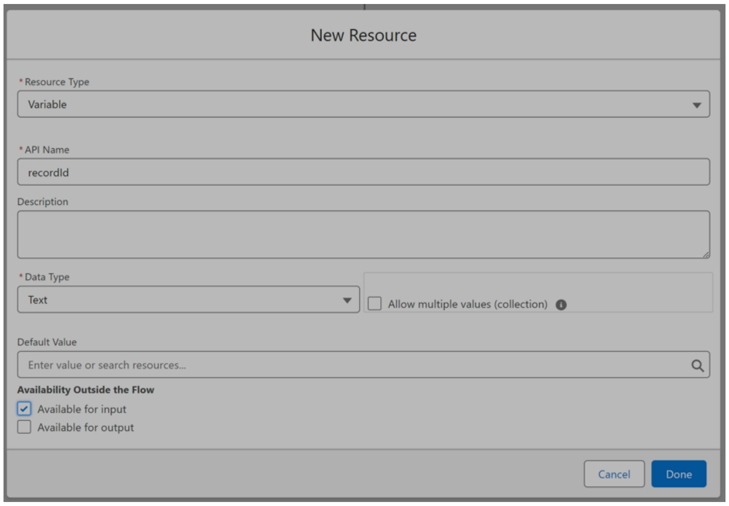
CPO Card Actions

CPO Card Actions support running:

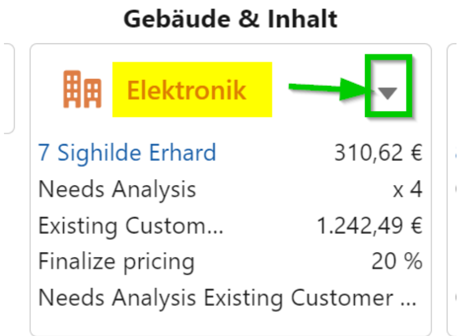
- Screen Flows
- Auto-lunched Flows (under development and planned to be delivered in next release)

Screen Flow prerequisites

1. To support CPO Card actions, existing or newly created Screen Flow must have **String type:** `recordId` variable defined



which will be filled by CPO App with **record ID** behind CPO Card (link from title of the card)



2. CPO App does not come with any Screen Flow therefore to use CPO Card Actions, screen flows must be **created and activated** on org and can be started from CPO App via configured CPO Card Actions.

3. To run specific flow make sure to grant user access to it

By default, users with any of the following permissions can run this flow.

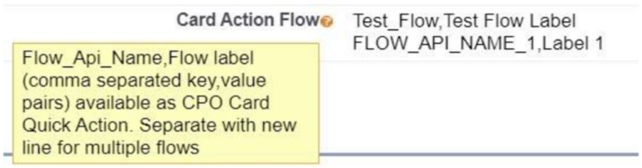
- Run Flow or Manage Flow in their profile or permission set
- Flow User in their user record

☐ Override default behavior and restrict access to enabled profiles or permission sets.

Recommended is to add all needed flows into CPO Permission Sets: CPO Admin / CPO User

4. Custom metadata configuration

Custom Metadata Type	Field
`neo_cpo__CPO_ReadingPathSetting__mdt`	`neo_cpo__CardActionFlow__c`



a. Provide API Names with labels in this field with **coma separated key value pairs**:
`API_NAME_1,Label 1`

b. For multiple actions, separate coma separated key value pars **with new lines**:

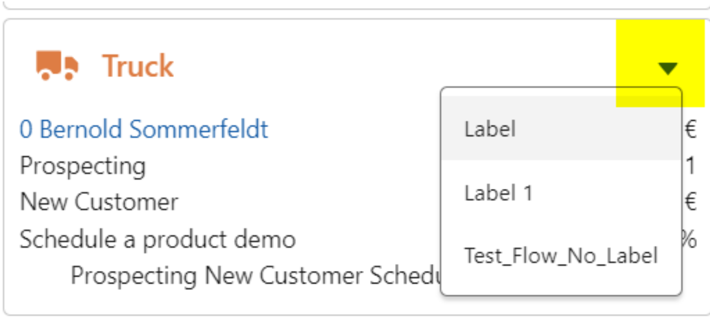
```
My_Awsome_Flow,Label no need to match with real flow label. Can be customized
here ANOTHER_API_NAME_2,Any Label here but recomended is to use the same like
flow label ONE_MORE_API_NAME_3
```

This will activate on all CPO Cards related to the specific Reading Path where the field was populated, dedicated CPO Card Actions.

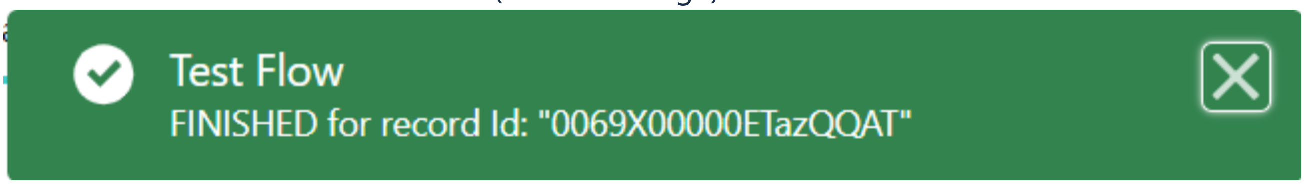
c. Api Name is **obligatory**.

d. Label is **not obligatory**. If not provided, Api Name will be used only.

5. Running CPO Card Actions
   - Click on the arrow icon in top right of CPO Card and simply select from list an action to run:



   - Validate that flow was successful (Toast message)

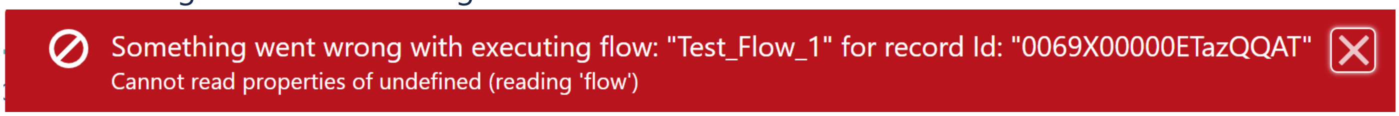


6. Error reporting

Make sure to provide valid **API Names** of the **Active** flows otherwise user will get error

Flow "Test_Flow_1" is not found or doesn't have an active version. Contact your admin for more information.

and following Toast Error Message:



Above Toast Error Message will also be shown in case flow was not finished successfully .

Enhanced Customization for NEO.CPO

The App does provide a more powerful degree of customization.

By respecting certain aspects you can create a full custom solution for the

- Partner Feature
- Persona Feature
- Reading Paths

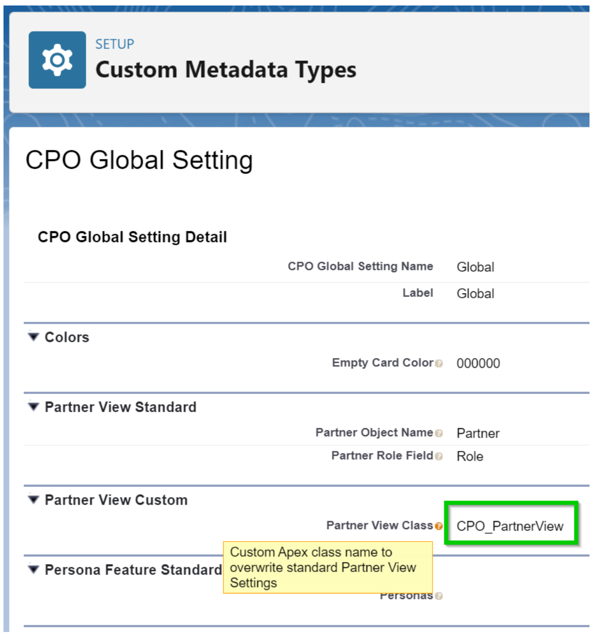
Custom Partner Feature

A fully customized class would require to call some methods from the App. Therefore we are providing an example that uses the Account field Parent for the partner feature:

```
global with sharing class CPO_Partner implements Callable {

    global List<neo_cpo.PartnerWrapper> call(String param, Map<String, Object> args) {
        Set<String> accIds = args.keySet(); // Account Ids to retrieve partners from database
        List<neo_cpo.PartnerWrapper> partnerWrapperList = getData(accIds);
        return partnerWrapperList;
    }


    private List<neo_cpo.PartnerWrapper> getData(Set<String> accIds) {
        //get Accounts (also named as rootAccounts of CPO context from database via bulkifyied mode
        //List<Account> rootAccounts = [SELECT Id, Name, RecordType.Name, RecordTypeId, ParentId, Parent.Name, Parent.RecordType.Name FROM Account WHERE
ID IN :accIds]; //-> use this if custom persona feature (with our example) is activated
        List<Account> rootAccounts = [SELECT Id, Name, Type, ParentId, Parent.Name, Parent.Type FROM Account WHERE ID IN :accIds]; //-> else use this

        //the custom CPO Partner example is about to get children and parent of rootAccount

        //get the childron of the rootAccounts
        //List<Account> childAccounts = [SELECT Id, Name, RecordType.Name, RecordTypeId, ParentId FROM Account WHERE ParentId IN :accIds]; //-> use this
if custom persona feature (with our example) is activated
        List<Account> childAccounts = [SELECT Id, Name, Type, ParentId FROM Account WHERE ParentId IN :accIds]; //-> else use this

        List<neo_cpo.PartnerWrapper> pwList = new List<neo_cpo.PartnerWrapper>();

        //loop the rootAccounts to build partner view
        for (Account acc : rootAccounts) {
            neo_cpo.PartnerWrapper pw = new neo_cpo.PartnerWrapper();

            //accFrom is always the root Account of the CPO context
            pw.accFromId = acc.Id;
            pw.accFromName = acc.Name;

            //provide persona information to partnerObj if you use the persona feature - do not provide if you do not use
            //pw.rootPersona = acc.RecordType.Name; //-> use this if custom persona feature (with our example) is activated
            //pw.rootPersona = acc.Type; //-> use this if standard persona feature is activated with field type

            //get children as a partner
            pw.partners = getChildren(acc.Id, childAccounts);

            //add the parent as a partner
            if(acc.ParentId != null) {

                neo_cpo.PartnerWrapper.PartnerObj partnerObj = new neo_cpo.PartnerWrapper.PartnerObj();
                partnerObj.id = acc.ParentId;
                partnerObj.name = acc.Parent.Name;
                partnerObj.role = 'Parent';

                //provide persona information to partnerObj if you use the persona feature - do not provide if you do not use
                //partnerObj.partnerPersona = acc.Parent.RecordType.Name; //-> use this if custom persona feature (with our example) is activated
                //partnerObj.partnerPersona = acc.Parent.Type; //-> use this if standard persona feature is activated with field type

                pw.partners.add(partnerObj);

            }

            pwList.add(pw);

        }
        return pwList;
    }

    private List<neo_cpo.PartnerWrapper.PartnerObj> getChildren(Id rootAccId, List<Account> children) {
```

```
        List<neo_cpo.PartnerWrapper.PartnerObj> partnersList = new List<neo_cpo.PartnerWrapper.PartnerObj>();

        //loop the children Accounts to find to add where rootAccount is parent
        for (Account childAcc : children) {
            if (childAcc.parentId == rootAccId) {
                neo_cpo.PartnerWrapper.PartnerObj partnerObj = new neo_cpo.PartnerWrapper.PartnerObj();
                partnerObj.id = childAcc.Id;
                partnerObj.name = childAcc.Name;
                partnerObj.role = 'Child';

                //provide persona information to partnerObj if you use the persona feature - do not provide if you do not use
                //partnerObj.partnerPersona = childAcc.RecordType.Name; //-> use this if custom persona feature (with our example) is activated
                //partnerObj.partnerPersona = childAcc.Type; //-> use this if standard persona feature is activated with field type

                partnersList.add(partnerObj);
            }
        }

        return partnersList;
    }
}
```

Note: You still need to populate class name in the global settings.



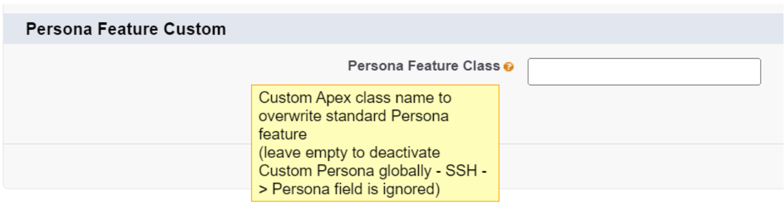
Custom Persona Feature

A fully customized class would require to call some methods from the App. Therefore we are providing an example that uses the Account Record Type to differ CPO dashboard views:

```
global with sharing class CPO_Persona implements Callable {

    global Object call(String param, Map<String, Object> accIds) {
        // param == 'CPO_PERSONA_CUSTOM'
        List<Account> accountsToGetPersonas = [
            SELECT RecordType.Name // <- sample persona field
            FROM Account
            WHERE Id IN :accIds.keySet()
        ];
        List<String> personas = new List<String>();
        for (Account acc : accountsToGetPersonas) {
            personas.add(acc.RecordType.Name);

        }
        return personas;
    }
```

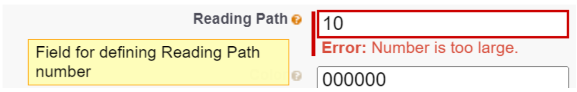
```
}
```

Note: You still need to populate class name in the global settings.



Custom Reading Path

NEO CPO application supports up to **9 Active Reading Paths (RP)**. It is limited via configuration schema **Reading Path** field. Validation will not allow entering number above MAX 9.



If standard reading paths are not enough, it is possible to configure custom reading paths via custom Apex classes. Both standard and custom RP can work simultaneously.

Custom Reading Path Considerations:

SOQL

To render date or number fields in NEO CPO App v2 on Cards in user specific locale, use in SOQL query:

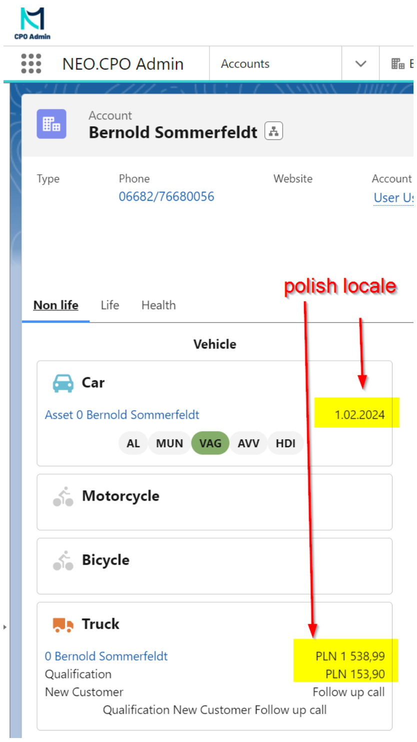
```
SELECT
    FORMAT(DateField) AliasForDateField,
    FORMAT(NumberFieldName) AliasForNumberFieldName,
    ...
```

https://developer.salesforce.com/docs/atlas.en-us.soql_sosl.meta/soql_sosl/sforce_api_calls_soql_select_format.htm

In Organizations with multicurrency, to render currency field in NEO CPO App v2 on Cards in user specific locale, use in SOQL query.

```
SELECT
    FORMAT(convertCurrency(CurrencyField)) AliasForCurrencyField,
    ...
```

https://developer.salesforce.com/docs/atlas.en-us.soql_sosl.meta/soql_sosl/sforce_api_calls_soql_querying_currency_fields.htm

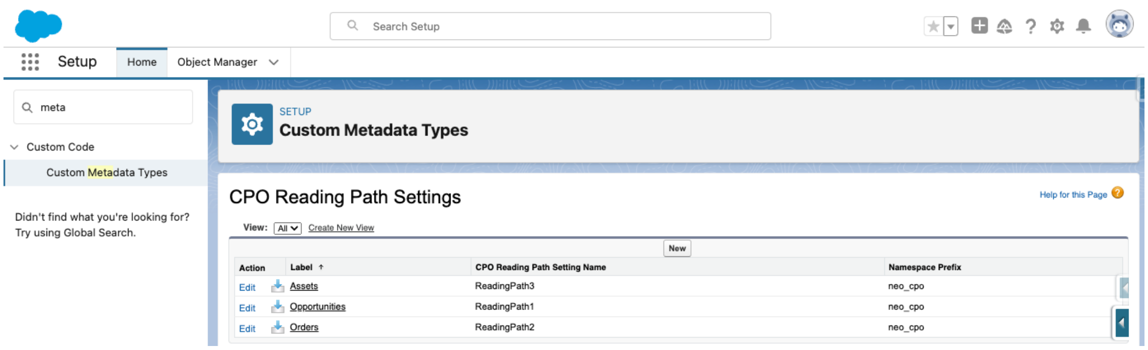


For a custom reading path you need to consider to call some App methods. Therefore we are providing a class example for a custom reading path with mocked data.

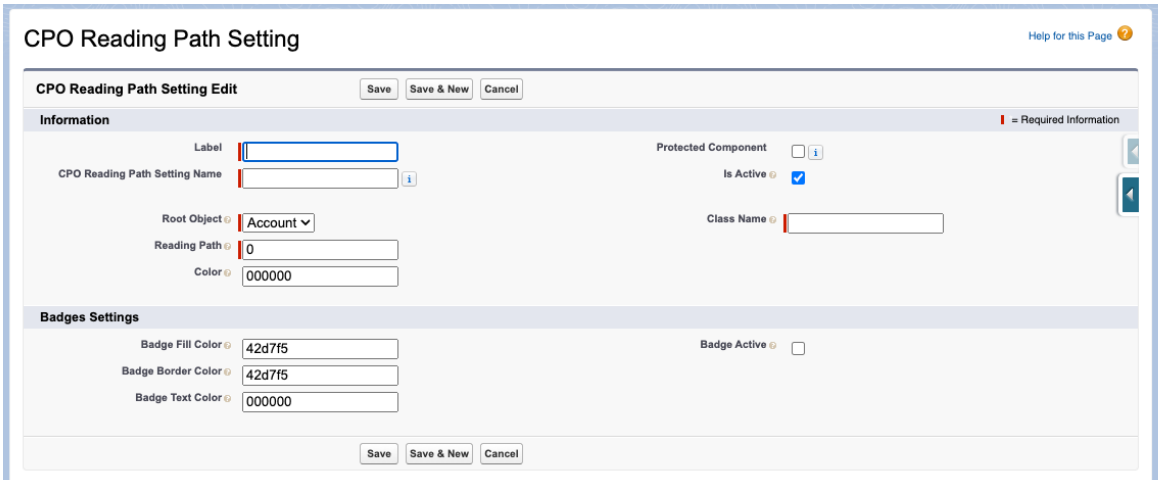
```
global class CPO_CustomRP implements Callable {
    public static final String VALID_PATH_NUMBER = '4', VALID_ROOT_OBJ_NAME = 'Account';
    private static final Set<Id> accIds = new Set<Id>();
    private static final Map<Id, neo_cpo__Sales_Service_Hierarchy__c> sshById = new Map<Id, neo_cpo__Sales_Service_Hierarchy__c>();

    public Object call(String pathNumber, Map<String, Object> args) {
        neo_cpo.ReadingPathHelper.validate(pathNumber, args, VALID_PATH_NUMBER, VALID_ROOT_OBJ_NAME);
        neo_cpo.ReadingPathHelper.extractArguments(args, VALID_ROOT_OBJ_NAME, accIds, sshById); //validated: accIds (rootAccount + partnerAccount) & sshById properties filled here
        return getCards();
    }

    private static List<neo_cpo.CpoWrapper.Card> getCards( ) {
        List<Case> data = getData();
        List<neo_cpo.CpoWrapper.Card> cards = new List<neo_cpo.CpoWrapper.Card>();
        for (Case cs : data) {
            cards.add(buildCard(cs.AccountId, cs));
        }
        return cards;
    }

    private static List<SObject> getData( ) {

        // Mocked Case data here for demo purpose but should be retrieved from org for all 'accIds'
        // both standard and custom objects are supported

        List<Case> data = new List<Case>();

        for ( id accId : accIds) {

            for (neo_cpo__Sales_Service_Hierarchy__c crdSsh : neo_cpo.UtilSelector.cardsLevel3Sshs.values()) {
                Boolean rnd = Math.mod(Integer.valueOf(Math.floor(Math.random() * 10)), 6) == 0;
                if (rnd) {
                    Case newCase = new Case(AccountId = accId, Description = crdSsh.Name, SuppliedName = crdSsh.Id, Comments = 'Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam');
                    data.add(newCase);
                    data.add(newCase.clone());
                }
            }
```

```
        }
        return data;
    }

    private static neo_cpo.CpoWrapper.Card buildCard(Id accId, SObject sObj) {
        // specify product - cardSsh lookup
        String crdSshId = (String) sObj?.get('SuppliedName');
        // use package method to build initial card as a mandatory step
        neo_cpo.CpoWrapper.Card card = neo_cpo.ReadingPathHelper.buildInitialCard(accId, crdSshId, sshById, Integer.valueOf(VALID_PATH_NUMBER));

        // fill up to 1-12 custom fields on card:
        card.custom1 = new neo_cpo.CpoWrapper.CustomField('Account ' + accId, 'Account', (String) accId); // -> use package CustomField constructor to
build an internal link with (label, objectName, id)
        card.custom2 = (String) sObj?.get('Description');
        card.custom3 = 'Custom 3';
        card.custom4 = 'Custom 4';
        card.custom5 = new neo_cpo.CpoWrapper.CustomField('Google Link','http://www.google.de'); // -> use package CustomField constructor to build an
external link with (label, url)
        card.custom6 = 'Custom 6';
        card.custom7 = 'Custom 7';
        card.custom8 = 'Custom 8';
        card.custom9 = (String) sObj?.get('Comments');
        card.custom10 = 'Custom 10';
        card.custom11 = 'Custom 11';
        card.custom12 = 'Custom 12';


        fillBadgesForCard(card);
        return card;
    }

    private static void fillBadgesForCard(neo_cpo.CpoWrapper.Card card) {
        if (card?.badges == null || card.badges.isEmpty()) {
            return;
        }
        Set<String> sshNames = neo_cpo.UtilSelector.badgesLevel4SshsByName.keySet();
        activateBadges(card);
    }

    private static void activateBadges(neo_cpo.CpoWrapper.Card card) {
        integer counter = 0;
        for (neo_cpo.CpoWrapper.Badge badge : card.badges) {
            counter++;
            if (counter > 1) {
                badge.active = true;
                badge.description = 'Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore
magna aliquyam';
            }
        }
    }

}
```

Note: You still need to create a new record in custom metadata settings.



Click New



All mandatory fields need to be populated with information based on the class that you have created.