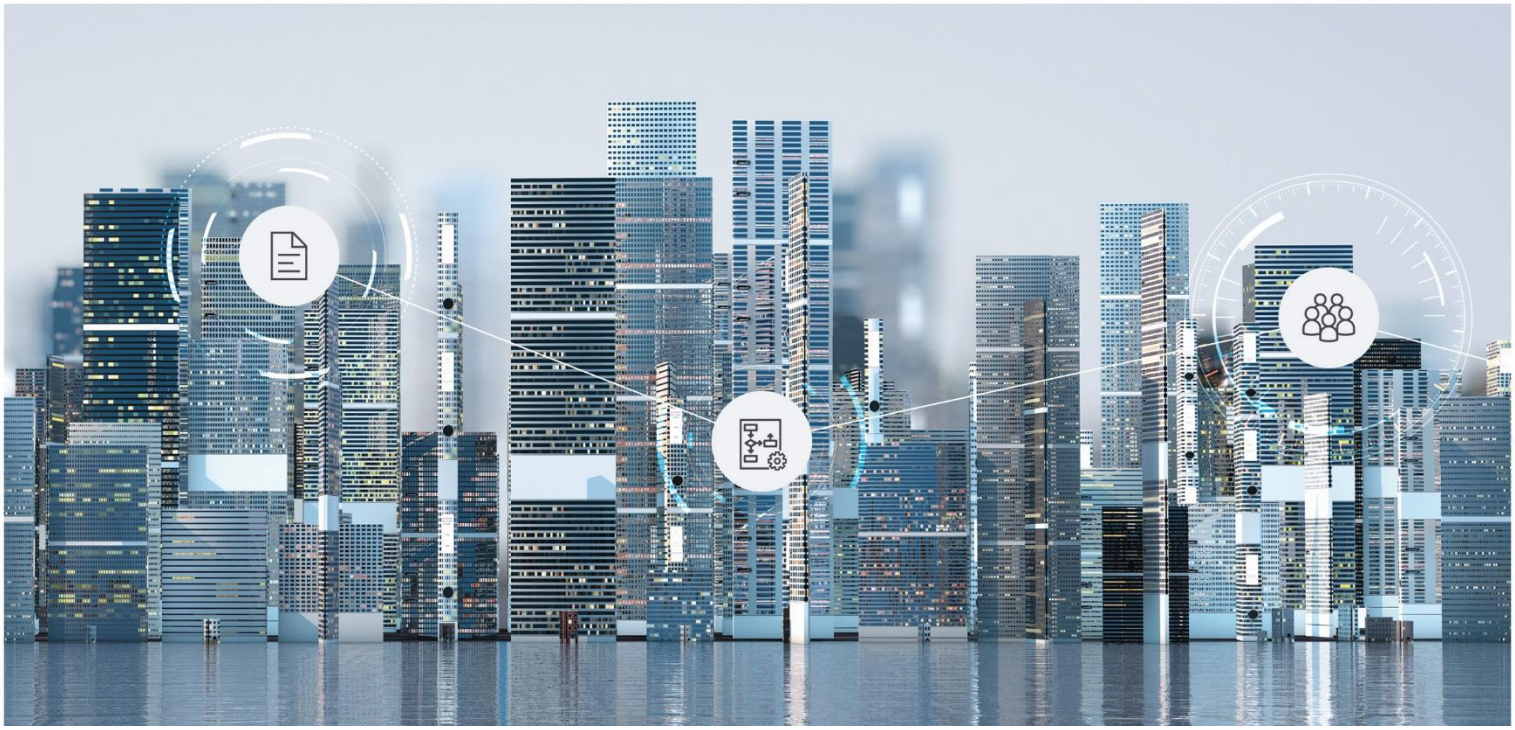
Fabasoft



White Paper

Developing Fabasoft Cloud Apps - Room Concept

2021 December Release

Contents

1 Introduction

The fundamental concept of the Fabasoft Cloud is to store data in rooms that are also used to grant access rights. The most common room is the Teamroom that allows exchanging documents and working together. As an app.ducx developer you can provide your own app room and app configuration for application-specific functionality. This document describes how to utilize app rooms and app configurations efficiently.

2 Prerequisites

It is assumed that you have some level of familiarity with developing Fabasoft Cloud apps. Essential information can be found here:

- https://help.cloud.fabasoft.com/index.php?topic=doc/Developing-Fabasoft-Cloud-Apps/index.htm
- https://help.appducx.fabasoft.com/index.php?topic=doc/An-Introduction-to-Fabasoft-appducx/index.htm

The app.ducx project that is discussed in this document can be downloaded here (your Fabasoft Cloud user name and password is required):

- https://folio.fabasoft.com/svn/public/Folio Cloud Apps/samples/democontractmgmt

3 Contract Management App

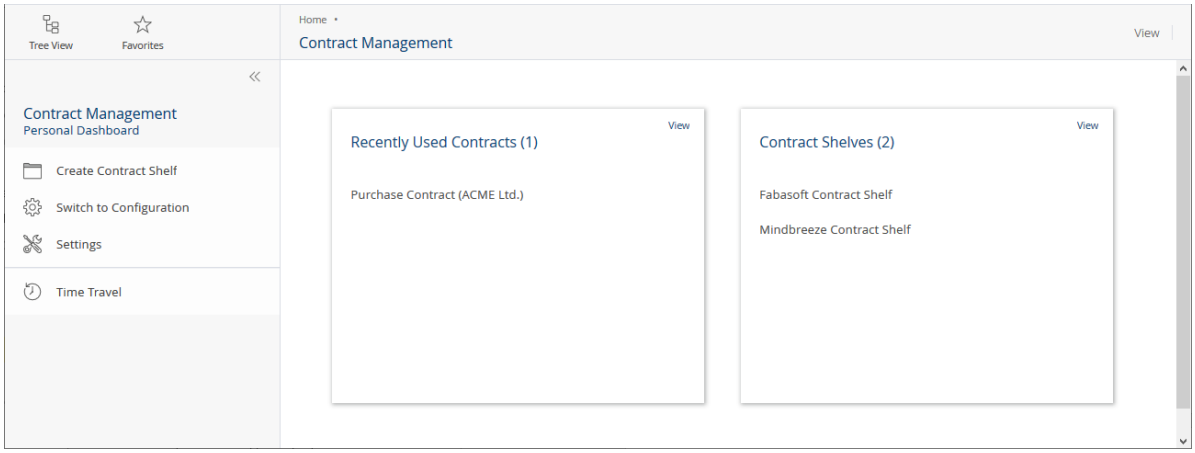
The contract management app example is provided in two stages:

- In the first stage, the contract management app room behaves like a Teamroom and contracts are modeled as compound objects.
  https://folio.fabasoft.com/svn/public/Folio Cloud Apps/samples/democontractmgmt/basic
- In the second stage, room roles are introduced, a chart is provided and contracts are modeled as files. In addition, some customizations are carried out to improve the user experience.
  https://folio.fabasoft.com/svn/public/Folio Cloud Apps/samples/democontractmgmt/advanced

Import the desired example (basic or advanced) in your Eclipse development environment.

4 Basic App Room

The contract management app should allow managing contracts in contract shelves (app rooms). Settings can be defined in the app configuration. The fully-implemented basic example looks like that:



4.1 Object Model

First of all, the app itself is needed.

app.ducx-om

```
objmodel DEMOCONTRACTMGMT@111.100
{
  import COOATTREDIT@1.1;
  import COODESK@1.1;
  import COOSYSTEM@1.1;
  import COOTC@1.1001;

  instance App AppContractMgmt {
    symbol = SymbolAgreement;
    appdescriptionstr = {}
  }
}
```

In addition, define the following object model:

- App Dashboard `ContractDashboard` (derived from `AppDashboard`)
  The dashboard is the central access point to your app. It provides a recently used list of contracts and a list of contract shelves (property `dbapprooms`).
- App Configuration `ContractConfiguration` (derived from `AppConfigurationRoom`)
  The configuration can be used to define customizations and store settings. `AppConfigurationRoom` provides by default fields like *Forms and Categories*, *Processes*, *Templates* and so on.
- App Room `ContractShelf` (derived from `AppRoom`)
  The room is used to store contracts and define access rights. `AppRoom` provides by default a Teamroom-like behavior.
- Contract `Contract` (derived from `CompoundObject`)
  The contract metadata and documents can be stored in contract objects.

model.ducx-om

```
objmodel DEMOCONTRACTMGMT@111.100
{
  import COOATTREDIT@1.1;
  import COODESK@1.1;
  import COOMAPI@1.1;
  import COOSYSTEM@1.1;
  import FSCFOLIO@1.1001;
  import FSCTEAMROOM@1.1001;

  /**
   * The dashboard for contract management
   */
  class ContractDashboard : AppDashboard {
    compapps = AppContractMgmt;
    symbol = SymbolAgreement;
    // Shows recently used contracts, for more information see the
    // description after this example
    unique Contract[] cdrecentlyusedcontracts volatile(tx) readonly {
      get = AttrCdRecentlyUsedContractsGet;
      copy = NoOperation;
    }
    // Helper property for the recently used list
    unique RecentlyUsedElements[] cdrecentlyusedelements invisible {
      attractsearch = AttrSearchNotPossible;
      copy = NoOperation;
    }
  }

  /**
   * The configuration room for the contract management
   */
  class ContractConfiguration : AppConfigurationRoom {
    compapps = AppContractMgmt;
    symbol = SymbolAgreement;
  }

  /**
   * The room containing all contracts
   */
  class ContractShelf : AppRoom {
    compapps = AppContractMgmt;
    symbol = SymbolRecordFilled;
    Contract[] cscontracts {
      child = true;
    }
  }

  /**
   * The contract containing all documents and meta data
   */
  class Contract : CompoundObject {
    compapps = AppContractMgmt;
    symbol = SymbolBill;
    string ctpartner not null;
    string[] ctdescription;
    date ctstart;
    date ctend {
      validate = expression {
        !::value || !cooobj.ctstart || cooobj.ctstart <= ::value;
      }
    }
    ContentObject[] ctdocuments {
      child = true;
    }
    Address[] ctaddress;
    Telephone[] cttelephone;
```

```
    EmailInformation[] ctemailinformation;
  }
}
```

To calculate the "Recently Used" list of the dashboard following is needed (see also `usecases.ducx-uc`):

- `cdrecentlyusedelements` is an invisible helper property that stores the recently used elements with time stamp.
- The actions `AddRecentlyUsedToAppDashboard` and `GetRecentlyUsedDisplayAttributes` need to be overridden to provide the "recently used" functionality for contracts.
- `cdrecentlyusedcontracts` shows the contracts based on the get action `AttrCdRecentlyUsedContractsGet`.
- The get action `AttrCdRecentlyUsedContractsGet` returns contracts based on the timestamp.

4.2 User Interface

Define the following user interface:

- The dashboard should show the last recently used list and the contract shelf list.
- The configuration should inherit default pages and show the contract shelf list.
- The contract shelf should show the contract list with defined columns.
- The contract should show the documents in explore mode and the metadata when editing the contract.
- Actions for creating a contract shelf or contract should be available to the user.

forms.ducx-ui

```
userinterface DEMOCONTRACTMGMT@111.100
{
  import COOATTREDIT@1.1;
  import COODESK@1.1;
  import COOSEARCH@1.1;
  import COOSYSTEM@1.1;
  import FSCTEAMROOM@1.1001;

  form FormContractDashboardExplore {
    inherit = false;
    formpage PageRecentlyUsedContracts {
      dataset { cdrecentlyusedcontracts; }
    }
    formpage PageDBContractShelves {
      dataset { dbapprooms; }
    }
  }

  form FormContractConfigurationExplore {
    inherit = true;
    formpage PageContractShelves {
      dataset { acapprooms; }
    }
  }

  form FormContractShelfExplore {
    inherit = false;
    formpage PageContracts {
      dataset { cscontracts; }
    }
```

```
}

form FromContract {
  formpage PageContract {
    dataset {
      Contract;
    }
    layout {
      row {
        objname {
          mustbedef = expression { true; }
        }
      }
      row { ctpartner; }
      row { ctdescription; }
      row {
        ctstart;
        ctend;
      }
      row { ctaddress; }
      row { cttelephone; }
      row { ctemailinformation; }
    }
  }
}

form FormContractExplore {
  inherit = false;
  formpage PageContractDocuments {
    dataset { ctdocuments; }
  }
}

forms for ContractDashboard {
   ExploreObject { FormContractDashboardExplore; }
}

forms for ContractConfiguration {
  ExploreObject { FormContractConfigurationExplore; }
}

forms for ContractShelf {
  ExploreObject { FormContractShelfExplore; }
}

forms for Contract {
  default, ObjectConstructor { FromContract; }
  ExploreObject { FormContractExplore; }
}

columns for ContractShelf {
  cscontracts {
    objname;
    ctpartner;
    ctstart;
    ctend;
  }
}

menus for ContractConfiguration {
  TaskPaneExpansion {
    target = [dashboard, acapprooms];
    MenuCreateContractShelf;
  }
}

menus for ContractDashboard {
```

```
    TaskPaneExpansion {
      target = [dashboard, dbapprooms];
      MenuCreateContractShelf;
    }
  }

  menus for ContractShelf {
    TaskPaneExpansion {
      MenuCreateContract;
    }
  }
}
```

4.3 Use Cases

Define the following functionality:

- The use cases for creating a contract shelf and a contract have to be implemented.
- The recently used functionality of the dashboard has to be implemented.

usecases.ducx-uc

```
usecases DEMOCONTRACTMGMT@111.100
{
  import COOATTREDIT@1.1;
  import COODESK@1.1;
  import COOSYSTEM@1.1;
  import COOTC@1.1001;
  import FSCFOLIO@1.1001;
  import FSCTEAMROOM@1.1001;
  import FSCVAPP@1.1001;
  import FSCVENV@1.1001;

  /**
   * Menu usecase to create a new contract shelf
   */
  menu usecase CreateContractShelf direct {
    symbol = SymbolRecordFilled;
    variant ContractConfiguration {
      application {
        expression {
          ->CreateObjectApp(cooobj, sys_action, #acapprooms, , , , true,
                            , , false, #ContractShelf, , #ContractShelf.
                            GetAttributeString(cootx, #mlname), , false,
                            true);
        }
        apphints = {
          MH_NEEDSCURRENTVERSION,
          MH_CREATESOBJ
        }
      }
    }
    variant ContractDashboard {
      application {
        expression {
          AppConfigurationRoom config = cooobj.GetAppConfiguration();
          if (config.IsUsable(#ContractConfiguration)) {
            ->CreateObjectApp(config, sys_action, #acapprooms, , , , true,
                              , , false, #ContractShelf, , #ContractShelf.
                              GetAttributeString(cootx, #mlname), , false,
                              true);
          }
        }
        apphints = {
```

```
        MH_NEEDSCURRENTVERSION,
        MH_CREATESOBJ
      }
    }
  }
}

/**
 * Menu usecase to create a new contract
 */
menu usecase CreateContract direct {
  symbol = SymbolAgreement;
  variant ContractShelf {
    application {
      expression {
        ->CreateObjectApp(cooobj, sys_action, #cscontracts, , , , true,
                          , , false, #Contract, , #Contract.
                          GetAttributeString(cootx, #mlname), , false,
                          true);
      }
      apphints = {
        MH_NEEDSCURRENTVERSION,
        MH_CREATESOBJ
      }
    }
  }
}

/**
 * Adds recently used contracts to the dashboard
 */
override AddRecentlyUsedToAppDashboard {
  variant Contract {
    expression {
      if (!cooobj.objistemplate) {
        ContractDashboard dashboard =
            coouser.GetUserDashboardByApp(#AppContractMgmt);
        if (dashboard.IsUsable(#ContractDashboard)) {
          try new transaction {
            dashboard.UpdateRecentlyUsedObjects(#cdrecentlyusedelements,
                                                #cdrecentlyusedcontracts,
                                                cooobj);
          }
        }
      }
    }
  }
}

/**
 * Retrieves the attribute containing recently used objects in the dashboard
 */
override GetRecentlyUsedDisplayAttributes {
  variant ContractDashboard {
    expression {
      attrdefs = #cdrecentlyusedcontracts;
    }
  }
}

/**
 * Calculate and sort the recently used contracts based on last usage
 */
AttrCdRecentlyUsedContractsGet(parameters as AttrGetPrototype) {
  variant ContractDashboard {
    expression {
      RecentlyUsedElements[] recentlyused =
```

```
            coouser.Sort(cooobj.cdrecentlyusedelements, false,
                         #ruetimestamp, false);
        value = unique(recentlyused.rueobject)[IsUsable(#Contract) &&
                                               !objistemplate];
      }
    }
  }
}
```

4.4 Customizations

Define the following customization:

- If the configuration does not exist, it should be automatically created when the license for the app is granted.

customization.ducx-cu

```
customization DEMOCONTRACTMGMT@111.100
{
  import COOATTREDIT@1.1;
  import COOSYSTEM@1.1;
  import FSCTEAMROOM@1.1001;

  target default {
    /**
     * Automatically create a ContractConfiguration after the license
     * for the app is granted
     */
    customize CPAutoCreateAppConfiguration<ContractConfiguration> {
      autocreate = true;
    }
  }
}
```

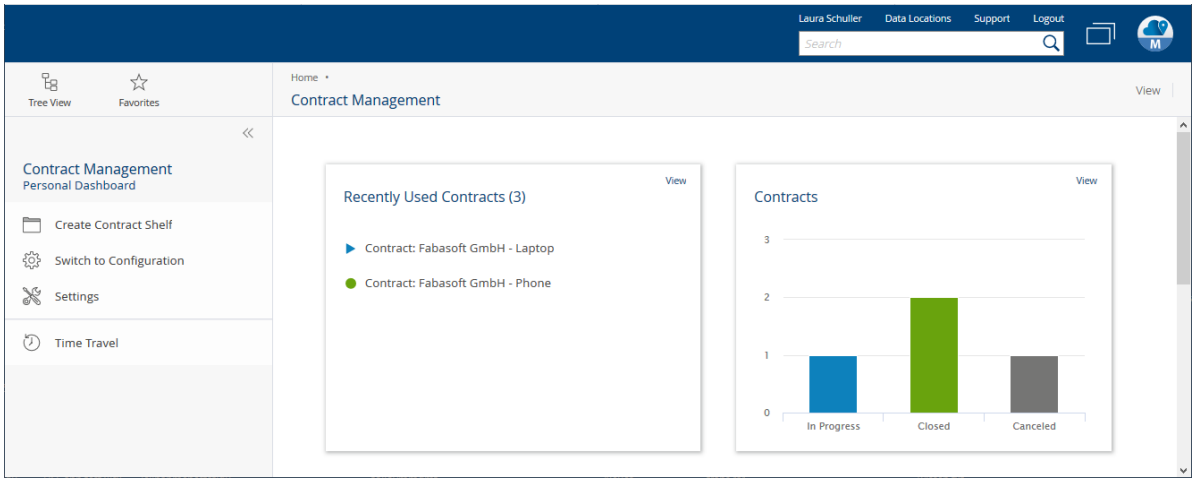
4.5 Result

Only the few steps depicted in the previous chapters are necessary to provide your own room for storing contracts. Load this example in your test environment and have a try.

5 Advanced App Room

In the following, the first example will be extended by more advanced functionality:

- Room Role
  The room role “Contract Inspection” is added to contract shelves to provide an own access right in addition to the default access rights (full control, change access and read access). The user may only see a contract, if the user is defined in the *Inspection* field of the contract.
- Contracts as Files
  Contracts are defined as files. In this way, typical file use cases (e.g. scan, cancel) are available out of the box.
- Microsoft Word Fields
  Contract metadata is available as Microsoft Word fields.
- Chart
  A chart visualizes the number of contracts based on their state.



5.1 Object Model

As described in the basic sample, the app itself is needed.

app.ducx-om

```
objmodel DEMOCONTRACTMGMT@111.100
{
  import COOATTREDIT@1.1;
  import COODESK@1.1;
  import COOSYSTEM@1.1;
  import COOTC@1.1001;

  instance App AppContractMgmt {
    symbol = SymbolAgreement;
    appdescriptionstr = {}
  }
}
```

Define the following object model. It is nearly the same as described in the basic sample but extended by some features.

- Struct for the Chart `ChartState`
  Extended functionality: Stores the state-amount-pair that is used to visualize the number of contracts based on their state.
- App Dashboard `ContractDashboard` (derived from `AppDashboard`)
  The dashboard is the central access point to your app. It provides a recently used list of contracts and a list of contract shelves.
  Extended functionality: In addition, a chart is provided.
- App Configuration `ContractConfiguration` (derived from `AppConfigurationRoom`)
  The configuration can be used to define customizations and store settings.
  `AppConfigurationRoom` provides by default fields like *Forms and Categories*, *Processes*, *Templates* and so on.
- App Room `ContractShelf` (derived from `AppRoom`)
  The room is used to store contracts and define access rights. `AppRoom` provides by default a Teamroom-like behavior.
- Contract `Contract` (derived from `CompoundObject`)
  The contract metadata and documents can be stored in contract objects.

Extended functionality: In addition, an inspection user can be stored and the contract metadata is available as Microsoft Word fields.

- Room Role `RoleContractInspection`
  Extended functionality: Users with this role may inspect their own contracts.
- Mindbreeze Search
  Extended functionality: All properties of the contracts should be searchable via Mindbreeze. This is necessary because not all properties of compound objects are available for search by default.

model.ducx-om

```
objmodel DEMOCONTRACTMGMT@111.100
{
  import COOATTREDIT@1.1;
  import COODESK@1.1;
  import COOMAPI@1.1;
  import COOSYSTEM@1.1;
  import FSCFIELDS@1.1001;
  import FSCFOLIO@1.1001;
  import FSCFOLIOCLOUD@1.1001;
  import FSCHIGHCHARTS@1.1001;
  import FSCMINDBREEZE@1.1001;
  import FSCTEAMROOM@1.1001;

  /**
   * Structure to display the states chart
   */
  struct ChartState {
    Object csstate;
    integer csamount;
    typeuniqueattrs = {
      csstate
    }
  }

  /**
   * The dashboard for contract management
   */
  class ContractDashboard : AppDashboard {
    compapps = AppContractMgmt;
    symbol = SymbolAgreement;
    unique Contract[] cdrecentlyusedcontracts volatile(tx) readonly {
      get = AttrCdRecentlyUsedContractsGet;
      copy = NoOperation;
    }
    unique RecentlyUsedElements[] cdrecentlyusedelements invisible {
      attractsearch = AttrSearchNotPossible;
      copy = NoOperation;
    }
    // The values are retrieved by the get action AttrCdStateChartGet
    // The control options define the layout
    ChartState[] cdstatechart volatile(tx) readonly {
      get = AttrCdStateChartGet;
      attrrepresentation<uiaction> = { { CTRLHighcharts } }
      visible = expression { cooobj.cdstatechart != null }
      controloptions = expression {
        return {
          chart: { type: 'column' },
          title: { text: '' },
          yAxis: {
            allowDecimals: false,
            min: 0,
            title: { text: '' }
```

```
        },
        xAxis: {
          title: { text: '' }
        },
        legend: { enabled: false },
        credits: { enabled: false },
        navigation: {
          buttonOptions: { enabled: false }
        },
        plotOptions: {
          column: {
            stacking: 'normal',
            format: '{series.name}'
          }
        },
        series: [
          {
            colorByPoint: true,
            colors: ['#007ab9', '#619e00', '#6e6e6d']
          }
        ]
      };
    }
  }
}

/**
 *  The configuration room for contract management
 */
class ContractConfiguration : AppConfigurationRoom {
  compapps = AppContractMgmt;
  symbol = SymbolAgreement;
}

/**
 * The room containing all contracts
 */
class ContractShelf : AppRoom {
  compapps = AppContractMgmt;
  symbol = SymbolRecordFilled;
  classdefaultacl = ContractShelfObjectsACL;
  Contract[] cscontracts {
    child = true;
    // Hide contracts without read access (otherwise users with
    // inspection role would see contracts without read access
    // as access denied entries)
    attractuifilter = AttrIsUsableUIFilter;
  }
  unique AdministrationObject[] csinspection {
    allow {
      User;
      Group;
    }
    accget = AccTypeReadSecRel;
    accset = AccTypeChangeSecRel;
    copy = NoOperation;
  }
}

/**
 * Contract containing all documents and meta data
 */
class Contract : CompoundObject {
  compapps = AppContractMgmt;
  symbol = SymbolBill;
  string ctitem not null;
  string ctpartner not null;
```

```
    string[] ctdescription;
    date ctstart;
    date ctend {
      validate = expression { !::value || !cooobj.ctstart ||
                              cooobj.ctstart <= ::value; }
    }
    ContentObject[] ctdocuments {
      child = true;
      changeable = expression { cooobj.bostate != #StateClosed }
    }
    // The user who should be able to inspect the own contract
    User ctinspection {
      allow {
        User;
      }
      accget = AccTypeReadSecRel;
      accset = AccTypeChangeSecRel;
      copy = NoOperation;
      filter = expression as attrfilterobjectexpr {
                 OBJECTLIST(cooobj.GetObjectRoom().GetTeamMembers()) }
    }
    /**
     * Properties available as Word fields
     */
    classglobalfields<fcfieldlist, component> = {
      {
        {
          ctitem,
          ctpartner,
          ctdescription,
          ctstart,
          ctend
        },
        DEMOCONTRACTMGMT@111.100
      }
    }
  }

  /**
   * Room role for the contract inspection
   */
  instance RoomRole RoleContractInspection {
    compapps = AppContractMgmt;
    roleattribute = csinspection;
    roleuireadonly = false;
    assignmentevent = ET_AddReadAccess;
    roleprioritу = RRP_USER_RESTRICTED;
  }

  /**
   * All contract properties should be searchable via Mindbreeze
   */
  extend instance MindbreezeDefaultConfig {
    fscmbobjectmappings<fscmbobjectclass, fscmballattrs,
                        fscmbsoftwarecomponent> = {
      { Contract, true, DEMOCONTRACTMGMT@111.100 }
    }
  }
}
```

5.2 User Interface

Define the following user interface:

- The dashboard should show the last recently used list and the contract shelf list.
  Extended functionality: In addition, the chart is shown.
- The configuration should inherit default pages and show the contract shelf list.
- The contract shelf should show the contract list with defined columns.
- The contract should show the documents in explore mode and the metadata when editing the contract.
- Actions for creating a contract shelf and contract should be available to the user.
  Extended functionality: The create contract shelf action is only visible if the user has the necessary access rights.
- **Extended functionality:** File use cases are available for contracts.

forms.ducx-ui

```
userinterface DEMOCONTRACTMGMT@111.100
{
  import COOATTREDIT@1.1;
  import COODESK@1.1;
  import COOSEARCH@1.1;
  import COOSYSTEM@1.1;
  import FSCFOLIO@1.1001;
  import FSCTEAMROOM@1.1001;

  form FormContractDashboardExplore {
    inherit = false;
    formpage PageRecentlyUsedContracts {
      dataset { cdrecentlyusedcontracts; }
    }
    formpage PageContractChart {
      dataset { cdstatechart; }
    }
    formpage PageDBContractShelves {
      dataset { dbapprooms; }
    }
  }

  form FormContractConfigurationExplore {
    inherit = true;
    formpage PageContractShelves {
      dataset { acapprooms; }
    }
  }

  form FormContractShelfExplore {
    inherit = false;
    formpage PageContracts {
      dataset { cscontracts; }
    }
  }

  form FromContract {
    formpage PageContract {
      dataset {
        Contract;
      }
      layout {
        row {
          objname {
```

```
          visible = expression { !cootx.IsCreated(cooobj) }
        }
      }
      row {
        ctitem;
        ctpartner;
      }
      row { ctinspection; }
      row { ctdescription; }
      row {
        ctstart;
        ctend;
      }
    }
  }
}

form FormContractExplore {
  inherit = false;
  formpage PageContractDocuments {
    dataset { ctdocuments; }
  }
}

forms for ContractDashboard {
   ExploreObject { FormContractDashboardExplore; }
}

forms for ContractConfiguration {
  ExploreObject { FormContractConfigurationExplore; }
}

forms for ContractShelf {
  ExploreObject { FormContractShelfExplore; }
}

forms for Contract {
  default, ObjectConstructor { FromContract; }
  ExploreObject { FormContractExplore; }
}

columns for ContractShelf {
  cscontracts {
    objname;
    ctpartner;
    ctstart;
    ctend;
  }
}

menus for ContractConfiguration {
  TaskPaneExpansion {
    target = [dashboard, acapprooms];
    MenuCreateContractShelf;
  }
}

menus for ContractDashboard {
  TaskPaneExpansion {
    target = [dashboard, dbapprooms];
    MenuCreateContractShelf;
    // Usability improvement: the menu is only shown if the user is
    // allowed to execute the command (otherwise an error message
    // would be shown)
    condition = expression {
      ContractConfiguration config = cooobj.GetAppConfiguration();
      config.IsUsable(#ContractConfiguration) &&
```

```
                        config.CheckSetAccess(cootx, #acapprooms);
      }
    }
  }

  menus for ContractShelf {
    TaskPaneExpansion {
      MenuCreateContract;
    }
  }

  menus for Contract {
    TaskPaneExpansion {
      MenuObjectImport;
      MenuScanFileObject;
      priority = 100;
      condition = expression { cooobj.bostate == #StateInProgress }
    }
    MenuContextExpansion  {
      MenuCloseContract;
      condition = expression { cooobj.bostate == #StateInProgress; }
    }
  }
}
```

5.3 Use Cases

Define the following functionality:

- The use cases for creating a contract shelf and a contract have to be implemented.
- **Extended functionality:** The use case for closing the contract has to be implemented.
- The recently used functionality of the dashboard has to be implemented.
- **Extended functionality:** The contract is defined as file. Consider special file use cases.
- **Extended functionality:** The chart values must be calculated.

usecases.ducx-uc

```
usecases DEMOCONTRACTMGMT@111.100
{
  import COOATTREDIT@1.1;
  import COODESK@1.1;
  import COOSYSTEM@1.1;
  import COOTC@1.1001;
  import FSCFOLIO@1.1001;
  import FSCTEAMROOM@1.1001;
  import FSCVAPP@1.1001;
  import FSCVENV@1.1001;

  /**
   * Menu usecase to create a new contract shelf
   */
  menu usecase CreateContractShelf direct {
    symbol = SymbolRecordFilled;
    variant ContractConfiguration {
      application {
        expression {
          ->CreateObjectApp(cooobj, sys_action, #acapprooms, , , , true,
                            , , false, #ContractShelf, ,
                            #ContractShelf.GetAttributeString(cootx,
                            #mlname), , false, true);
        }
        apphints = {
```

```
        MH_NEEDSCURRENTVERSION,
        MH_CREATESOBJ
      }
    }
  }
  variant ContractDashboard{
    application {
      expression {
        AppConfigurationRoom config = cooobj.GetAppConfiguration();
        if (config.IsUsable(#ContractConfiguration)) {
          ->CreateObjectApp(config, sys_action, #acapprooms, , , , true,
                            , , false, #ContractShelf, ,
                            #ContractShelf.GetAttributeString(cootx,
                            #mlname), , false, true);
        }
      }
      apphints = {
        MH_NEEDSCURRENTVERSION,
        MH_CREATESOBJ
      }
    }
  }
}

/**
 * Menu usecase to create a new contract
 */
menu usecase CreateContract direct {
  symbol = SymbolAgreement;
  variant ContractShelf {
    application {
      expression {
        ->CreateObjectApp(cooobj, sys_action, #cscontracts, , , , true,
                          , , false, #Contract, ,
                          #Contract.GetAttributeString(cootx,
                          #mlname), , false, true);
      }
      apphints = {
        MH_NEEDSCURRENTVERSION,
        MH_CREATESOBJ,
        MH_CHANGESVIEW
      }
    }
  }
}

/**
 * Closes the selected contract
 */
menu usecase CloseContract on selected {
  symbol = SymbolClose;
  variant Contract {
    application {
      expression {
        cooobj.ObjectLock(true, true);
        cooobj.bostate = #StateClosed;
        cooobj.ctdocuments.DoCloseFileObject();
      }
      apphints = {
        MH_CHANGESOBJ
      }
    }
  }
}

/**
 * Adds recently used contracts to the dashboard
```

```
 */
override AddRecentlyUsedToAppDashboard {
  variant Contract {
    expression {
      if (!cooobj.objistemplate) {
        ContractDashboard dashboard =
          coouser.GetUserDashboardByApp(#AppContractMgmt);
        if (dashboard.IsUsable(#ContractDashboard)) {
          try new transaction {
            dashboard.UpdateRecentlyUsedObjects(#cdrecentlyusedelements,
              #cdrecentlyusedcontracts, cooobj);
          }
        }
      }
    }
  }
}

/**
 * Retrieves the attribute containing recently used objects in the
 * dashboard
 */
override GetRecentlyUsedDisplayAttributes {
  variant ContractDashboard {
    expression {
      attrdefs = #cdrecentlyusedcontracts;
    }
  }
}

/**
 * Calculate and sort the recently used contracts based on last usage
 */
AttrCdRecentlyUsedContractsGet(parameters as AttrGetPrototype) {
  variant ContractDashboard {
    expression {
      RecentlyUsedElements[] recentlyused =
        coouser.Sort(cooobj.cdrecentlyusedelements, false, #ruetimestamp,
                     false);
      value = unique(recentlyused.rueobject)[IsUsable(#Contract) &&
                     !objistemplate && bostate != #StateCanceled];
    }
  }
}

/**
 * Define a contract as a file to enable the file use cases (Cancel,
 * Restore, Scan, ...) for the documents and remove the
 * unshare/delete menu
 */
override IsObjectFile {
  variant Contract {
    expression {
      isfile = true;
      enableusecases = true;
    }
  }
}

/**
 * Set the state to "In Progress" for new contracts
 */
override ObjectConstructor {
  variant Contract {
    expression {
      super();
      cooobj.bostate = #StateInProgress;
```

```
    }
  }
}

/**
 * Always unshare the contract from the contracts list of the contract
 * shelf in cancel use case
 */
override DoCancelFileObject {
  variant Contract {
    expression {
      parent = cooobj.GetObjectRoom();
      view = #cscontracts;
      super();
    }
  }
}

/**
 * Closed contract should not be editable in ui
 */
override IsEditable {
  variant Contract {
    expression {
      super();
      if (iseditable != false) {
        iseditable = cooobj.bostate != #StateClosed;
      }
    }
  }
}

/**
 * Calculates all original children of an Contract that should be moved
 * along, when moving/removing the contract to/from a wastebasket
 */
override GetOriginalChildren {
  variant Contract {
    expression {
      if (cooobj not in containersvisited) {
        containersvisited += cooobj;
        ContractShelf contractshelf = cooobj.GetObjectRoom();
        children *= cooobj.ctdocuments[!HasClass(#Room) &&
                                       GetObjectRoom() ==
                                       :>contractshelf &&
                                       GetLinkState(cooobj, #templates)
                                       == LT_ORIGINAL];
                                       //Ignore shortcuts
      }
    }
  }
}

/**
 * Get action used to calculate the highchart values to display the
 * amount of contracts per state
 */
AttrCdStateChartGet(parameters as AttrGetPrototype) {
  variant ContractDashboard {
    expression {
      assume ref ChartState[] value;
      value = null;
      if (!#TV.TV_RENDERTREE && !#TV.TV_GENERATE_ROOM_BREADCRUMB) {
        ContractShelf[] shelves =
          cooobj.dbapprooms[IsUsable(#ContractShelf)];
        if (shelves) {
          Contract[] contracts = [shelves.cscontracts,
```

```
            shelves.objcanceledfiles][IsUsable(#Contract)];
          if (contracts) {
            for (ComponentState state : [#StateInProgress, #StateClosed,
                 #StateCanceled]) {
              value += {
                csstate : state,
                csamount : 0
              };
            }
            for (Object contract : contracts) {
              ComponentState state = contract.bostate;
              if (value[csstate == :>state]) {
                value[csstate == :>state].csamount++;
              }
            }
          }
        }
      }
    }
  }
}
```

5.4 Customizations

Define the following customization:

- If the configuration does not exist, it should be automatically created when the license for the app is granted.
- **Extended functionality:** Consider the inspection role.
- **Extended functionality:** Hide the "Close" command for documents in contracts. In this case documents should only be closed when the file itself is closed (see also the menu use case `CloseContract`).
- **Extended functionality:** To optimize the ACL checks during query processing, the `SearchSecACLs` customization point can be used to define the `ContractShelfObjectsACL`.
- **Extended functionality:** Improve the visualization and usability.
- **Extended functionality:** Use the ACL that considers the inspection role.

For more information on room customization points, see also https://help.appducx.fabasoft.com/index.php?topic=doc/Predefined-Customization-Points/room.htm

customization.ducx-cu

```
customization DEMOCONTRACTMGMT@111.100
{
  import COOATTREDIT@1.1;
  import COODESK@1.1;
  import COOSYSTEM@1.1;
  import FSCCONFIG@1.1001;
  import FSCFOLIO@1.1001;
  import FSCFOLIOCLOUD@1.1001;
  import FSCFOLIOCLOUDPRECONFIG@1.1001;
  import FSCTEAMROOM@1.1001;
  import FSCVAPP@1.1001;

  target default {
    /**
     * Automatically create a ContractConfiguration after the license
```

```
 * for the app is granted
 */
customize CPAutoCreateAppConfiguration<ContractConfiguration> {
  autocreate = true;
}

/**
 * Add the inspection role to the room roles
 */
customize CPGetRoomRoles<ContractShelf> {
  roles = expression { [#RoleTeamRoomFullControl,
                        #RoleTeamRoomCanChange,
                        #RoleTeamRoomReadOnly,
                        #RoleContractInspection]; }
  default = expression { #RoleTeamRoomCanChange; }
  multipleroles = false;
}

/**
 * Replace the default quick search action with
 * FSCFOLIOCLOUDPRECONFIG@1.1001:RestrictedSearchInvitationRecipient
 * to improve the user/group calculation in team control
 */
customize CPQuickSearchAction<ContractShelf, csinspection> {
  cfgqsaction = RestrictedSearchInvitationRecipient;
}

/**
 * Improve appearance of users/groups in team control
 */
customize CPQuickSearchAppearance<ContractShelf, csinspection> {
  appearance = QS_ENHANCED;
}

/**
 * Hide file use case "Close" for documents in contracts
 */
customize CPGetObjFileUseCase<MenuCloseFileObject, ContentObject,
                               Contract, ContractShelf> {
  usecase = expression { null }
}

/**
 * Allow searching for objects with the ContractShelfObjectsACL ACL
 * via Mindbreeze
 */
customize SearchSecACLs<ContractShelfObjectsACL> {}


/**
 * Apply the ContractShelfObjectsACL ACL for all objects in
 * contract shelves
 */
customize CPGetRoomSecurity<Object, ContractShelf, CtxApplyRoom> {
  seccontext = expression {
    SecurityContext ({ objaclobj : #ContractShelfObjectsACL });
  }
}

/**
 * Allow changing the room assignment of an contract/document, that
 * is in multiple contract shelves
 * using the menu FSCTEAMROOM@1.1001:MenuChangeTeamRoom
 */
customize CPGetTargetRoomClasses<ContractShelf, Object,
                                 DEMOCONTRACTMGMT@111.100> {
  targetclasses = ContractShelf;
```

```
    }

    /**
     * Define contracts as containers considered in deletion use case,
     * so when a contract is moved to the wastebasket, the documents
     * will be moved along
     */
    customize CPGetWBContainers<ContractShelf> {
      containerclasses = expression { #Contract }
    }

    /**
     * When a user is excluded from the organization, replace the user
     * in the contract inspection property with his substitute
     */
    customize CPExcludeOrgMember<Contract, ContractShelf, null> {
      condition = expression { false }
      addsubstitute = expression { true };
      path<appath> = { { ctinspection } }
    }

    /**
     * Generate automatic name for contracts using the contractual item
     * and the contract partner
     */
    customize NameBuild<Contract> {
      properties = {
        ctitem,
        ctpartner
      }
      namefixed = true;
      build = expression { #Contract.Print() + ": " + cooobj.ctpartner +
                           " - " + cooobj.ctitem; }
    }

    /**
     * Show the state symbol next to the contract
     */
    customize CPStateDisplay<Contract> {
      cfgstatedisplayexpression = expression {
        ObjectStateDisplay[] dispstate = [];
        if (cooobj.bostate) {
          dispstate += {
            objectstatedisplaysymbol = cooobj.bostate.objmicon,
            objectstatedisplaydescription = cooobj.bostate.GetName()
          };
        }
        dispstate;
      }
    }
  }
}
```

5.5 ACL

Define the following ACL:

- **Extended functionality:** The ACL has to consider the inspection role and is assigned to all objects in contract shelves.

acls.ducx-os

```
orgmodel DEMOCONTRACTMGMT@111.100
{
  import COOSYSTEM@1.1;
```

```
  import COOWF@1.1;
  import FSCFOLIO@1.1001;
  import FSCTEAMROOM@1.1001;

  /**
   * ACL for contract shelves and contracts, that considers the
   * inspection role
   */
  acl ContractShelfObjectsACL {
    ace {
      audience = [TeamRoomOwnerShipAudience,
                  TeamRoomFullControlAudience];
      rights = [ TeamRoomRightsFullControl ];
    }
    ace {
      audience = [ { user objsecchange; } ];
      rights = [ TeamRoomRightsChangeAccess ];
    }
    ace {
      audience = [ { user objteamroom.objsecchange; } ];
      rights = [ TeamRoomRightsChildrenChangeAccess ];
    }
    ace {
      audience = [
        TeamRoomReadAccessAudience,
        { user objsecdelegated; },
        { user csinspection; },
        { user ctinspection; },
        { user objfile.ctinspection; }
      ];
     rights = [ TeamRoomRightsReadAccess ];
    }
    ace {
      audience = [ WorkFlowParticipantsAudience,
WorkFlowParticipantGroupsAudience ];
      rights = [ TeamRoomRightsWorkFlow ];
    }
  }
}
```

5.6 Result

The advanced example provided an overview of customizations that provide more functionality and enhance the user experience. Load this example in your test environment and have a try.