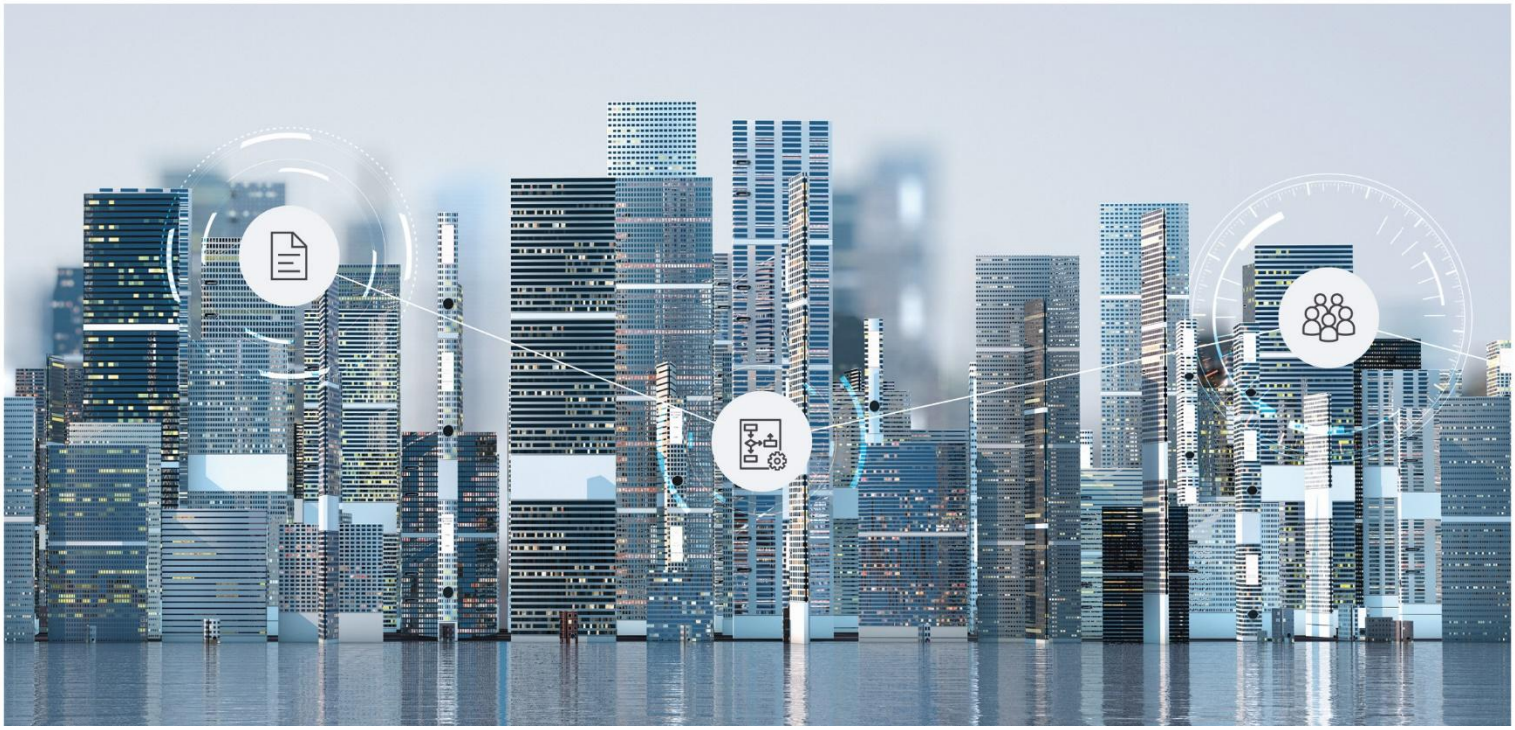
Fabasoft



White Paper

Model-Based Customizing

2025 July Release

Contents

1 Introduction

You can adapt the Fabasoft Cloud to your requirements by means of model-based customizing. This document outlines several scenarios in order to describe the options available to you. In the last chapter, you will find a comprehensive example.

2 Customization Options

Here you find a brief overview of the customization options available to you. All options are described in detail in the following chapters.

Forms

User-defined forms can be used to add fields to objects for storing application-specific data. To apply logic or specific behaviors to the fields, Fabasoft app.ducx expressions can be used.

Categories

Categories can be assigned to objects and thus influence the behavior of the objects.

Processes

Processes can be individually defined, and reflect your business and organizational structure. BPMN process diagrams are used to design executable business processes. Besides predefined BPMN processes, ad hoc processes can be started as needed.

Templates

Nearly all objects, and especially documents, can be defined as templates. In this way new, objects or documents based on a template can be created.

Text Modules

You can use text modules to insert predefined standard texts into Microsoft Word documents.

Display Settings

Predefined display settings can be provided to users who need special views on lists.

Search Forms

Predefined search forms can be provided to users who need an overview of currently existing objects based on defined search criteria.

Inboxes

For an inbox, rules for processing incoming objects can be defined. A rule consists of conditions and actions. This way, customer documents can be processed and distributed automatically.

3 Customization Scopes

The management of customizations is done on different levels (scopes). The different scopes are:

- Organization
  Organization-wide customizations are made in the customizing area, which is directly

accessible via the “Templates and Presettings” dashboard on “Home”.
All customizations can be defined organization-wide.

- App
  Apps like “Digital Asset Management” can provide their own customizations which are only available within the app configuration scope.
  The number of available customizations depends on the app.
- Room
  Rooms can provide their own customizations, which are only available within the room scope.
- User
  User-specific customization can only be applied by the user.

4 Release for Usage

To define which customization should be available to a user, the user has to be defined as a team member in the corresponding room that provides the customization.

In addition, the customization has to be released (“Release for Usage” action). A corresponding status symbol is displayed for not released customizing objects.

If the customization is changed, it has to be released again for the changes to take effect (“Re-Release” action). A corresponding status symbol is displayed for changed customizing objects. In the case of compound customizing objects, a status symbol is only displayed if the root object has changed.

If you do not want the customization to be available any longer, it can be withdrawn (“Withdraw Release” action).

5 Prerequisites

It is assumed that you have some level of familiarity with the topics outlined in chapter 2 “Customization Options”. More in-depth information about these topics can be found in the user help:

https://help.cloud.fabasoft.com/index.php?topic=doc/User-Help-Fabasoft-Cloud-eng/index.htm

In some cases, customizations require the use of Fabasoft app.ducx expressions. General information about Fabasoft app.ducx expressions can be found in the following document. Note that the document is intended for solution development, so not all concepts described are applicable for the purpose of customization.

https://help.cloud.fabasoft.com/index.php?topic=doc/Fabasoft-appducx-Expressions/index.htm

The following hints regarding Fabasoft app.ducx expressions may be helpful:

- Predefined values that can be accessed during the evaluation of an expression are provided in the local scope (`this`) or global scope (`::this`). Consult the reference documentation to find out which values are available in these scopes. The following document provides a link to the corresponding reference documentation for each expression property:.
  https://help.cloud.fabasoft.com/index.php?topic=doc/Reference-Documentation/customprops-overview.htm
- In most cases, a scope is either an object or a dictionary (key value pairs).

- In most cases, the keyword `this` can be omitted. If `this` is an object, `this.objname` is the same as `objname`.
- The software component `COOSYSTEM@1.1` can be omitted. `COOSYSTEM@1.1:objname` is the same as `objname`.
- If you want to use the short reference for other software components than `COOSYSTEM@1.1` as well, you can use the `import` keyword (e.g. `import FSCUSERFORMS@1.1001;`).
- If you need to refer to a component object, you must prefix its reference with `#` (e.g. `#objname`).
- To declare your own variables for necessary computations, the temporary scope (`@`) is available (e.g. `@availability = 2).`
- If you define a compound type you can use the programming name of the corresponding property to define the type of a variable (for example, in the *Expression for Computing the Value* field).
  ```
  myaggrlist[] @result = [];
  ```
  instead of
  ```
  FormAggregate_2_3505[] @result = [];
  ```
- `CheckGetSecured` and `CheckSetSecured` can be used to check secured get or set access for a property of an object (e.g. useful for user properties).
  Example:
  ```
  User @customer = cooobj.[#customer];
  if (@customer.CheckGetSecured(#usersurname)) {
    @customer.usersurname;
  }
  else {
    "Unknown";
  }
  ```
- Only properties and actions that are tagged as secure can be used in the expressions described in this document. An overview of secure properties and actions can be found here:
  https://help.cloud.fabasoft.com/index.php?topic=doc/Reference-Documentation/index.htm
- The code editor shows whether the language server that validates the entered expression is available (green dot).

When you use expressions for calculation or validation, it can sometimes be difficult to identify errors in the expressions. To simplify the analysis, you can write trace output to the web browser console. To do this, you must go to the context menu of the Teamroom in which the object is used, choose “Tools” > “Activate Trace Outputs” and allow trace output.

Call in expressions:

- `cooobj.Trace("string");`
- `cooobj.Trace("string", value);`

Output:

The output is a JSON data record.

- c
  Context of the call (Teamroom).
- d
  Current time.
- s
  Section of the expression (if available).

- t
  The text to be traced (first parameter of the trace call).
- u
  Current user.
- v
  The value to be traced (second parameter of the trace call).

6 Forms

In the graphical form editor, you can easily create new forms using drag-and-drop and extend objects with application-specific properties without requiring any programming knowledge. You can either add a form to an existing object (shown in the properties of the object), or you can directly create objects based on a form.

Useful for Following Tasks

- Objects and documents with customized data fields should be available.
- Rooms with customized data fields should be available (behave like Teamrooms).
- Containers with customized data fields should be available (may also be defined as file).
- Existing objects should be extended by customized data fields.
- Charts based on user-defined compound types should be available.

Essential information about forms can be found in the user help:

https://help.cloud.fabasoft.com/index.php?topic=doc/User-Help-Fabasoft-Cloud-eng/advanced-use-cases.htm#forms

6.1 Scenario

Objective

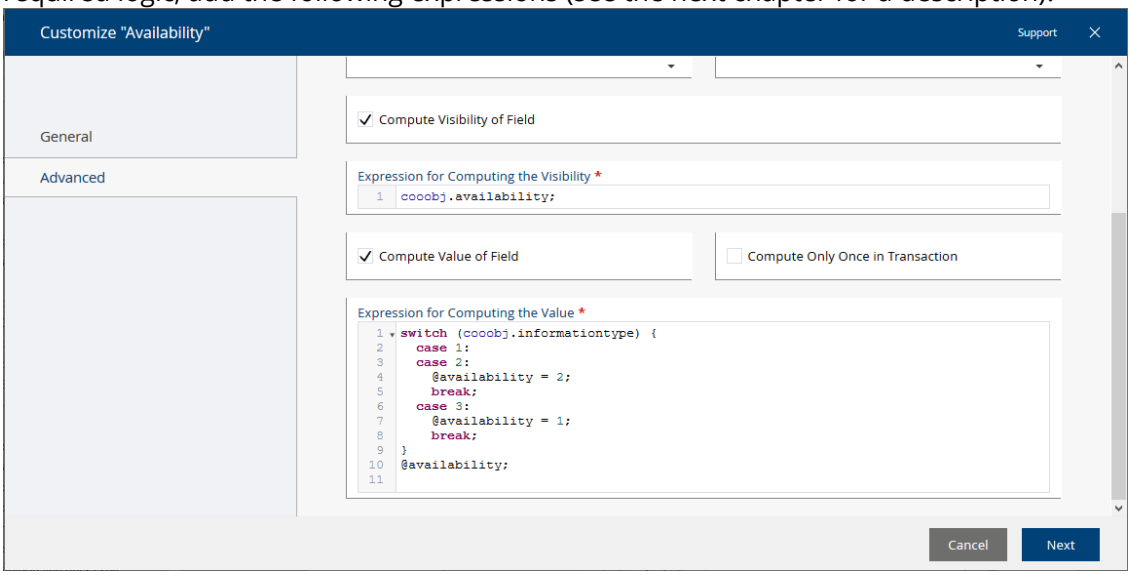
The “Development” department needs to structure their technical documents based on their own classification scheme. The following metadata should be available:

- Manufacturer
- Product
- Information Type (“User Manual”, “Service Manual”, “Specification”)
- Availability (“Internal”, “External”; read-only field based on the information type; not visible if no information type is defined)

To achieve the objective, proceed as follows:

- Navigate to “Templates and Presettings” > “Form and Category Collections” > “Your desired form collection”.
- Create a new form.
- Use the *Suppress Template Creation* option, to prevent objects from being created based on the form. The form should only be applied to existing documents.
- *Manufacturer* and *Product* can be defined as input fields of type string.

- The *Information Type* can be defined as a combo box with the required three values. The *Programming Name* is defined as “informationtype”. Add the following expression so that the availability is directly refreshed when the information type is changed:
  - *Handle Changes of Values* > *Expression for Handling Changes of Values*
    ```
    true;
    ```
- The *Availability* can be defined as combo box with the required two values. To apply the required logic, add the following expressions (see the next chapter for a description):



  - *Compute Visibility of Field* > *Expression for Computing the Visibility*
    ```
    cooobj.availability;
    ```
  - *Compute Value of Field* > *Expression for Computing the Value*
    ```
    switch (cooobj.informationtype) {
      case 1:
      case 2:
        @availability = 2;
        break;
      case 3:
        @availability = 1;
        break;
    }
    @availability;
    ```
- The form can be added to existing documents by assigning the corresponding *Category (Published)* to the documents (“General” tab > *Category*). The category can also be used to define additional settings (see chapter 7 “Categories”).
  Note: The settings of the *Category (Draft)* are applied to the *Category (Published)* when the form is (re-)released.

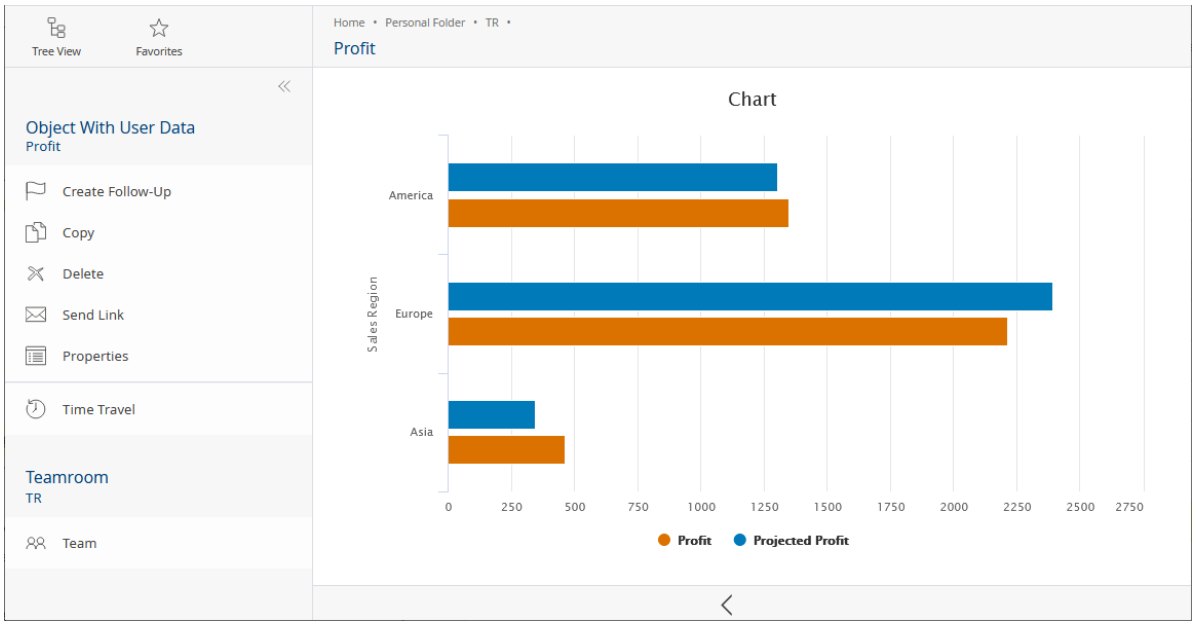
Objective

The “Sales” department needs to compare the profit with the projected profit for each sales region. In addition, a chart is needed to visualize the number of contracts based on their states.

To achieve the objective, proceed as follows:

Visualize the Profit for Each Sales Region

- Navigate to “Templates and Presettings” > “Form and Category Collections” > “Your desired form collection”.
- Create a new form with “Compound Type” as *Base Class*, and define three input fields:
  - Sales Region (*Type*: “String”)
  - Projected Profit (*Type*: “Float”)
  - Profit (*Type*: “Float”)
- Create a new form with “Compound Type” as *Base Class*, and define two item lists:
  - Values (*Type*: “Compound Type”, *Compound Type User Form*: the previously defined compound type, *Compound Type Display Mode*: “Standard”, *Programming Name*: “plainvalues”)
  - Chart (*Type*: “Compound Type”, *Compound Type User Form*: the previously defined compound type, *Compound Type Display Mode*: “Chart”, *Expression for Computing the Value*: `value = cooobj.plainvalues`, *Expression for Calculating the Control Options*: `dictionary({ chart: { type: "bar" } });`)
    The values for the chart are taken from the *Values* field using the expression for computing the value. The chart should be a bar chart.



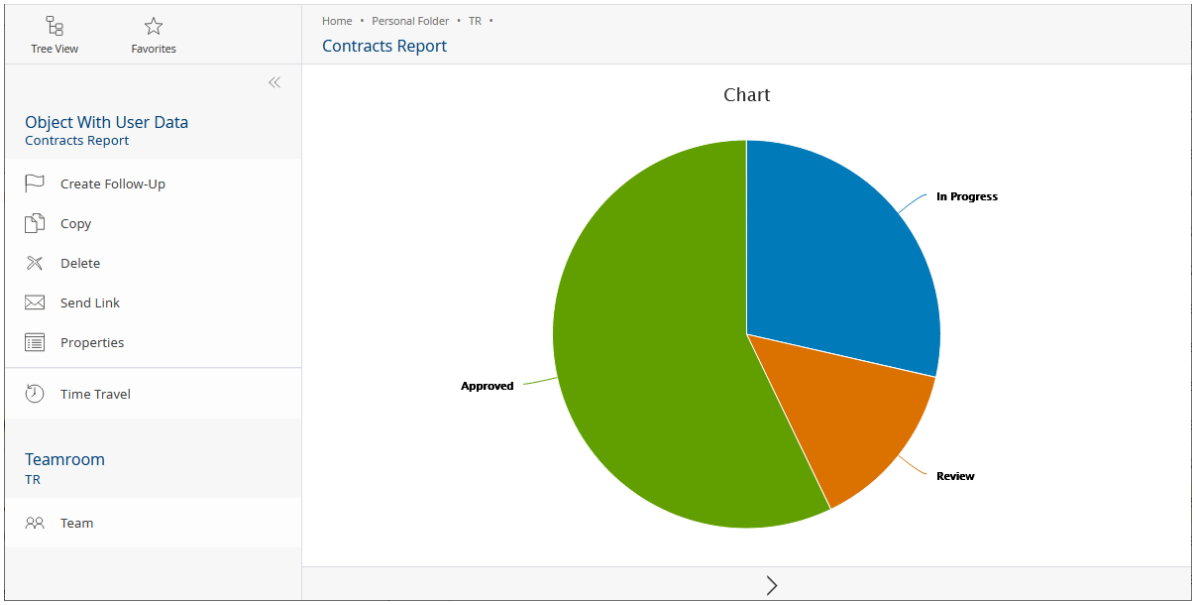
Visualize Contracts Based on Their State

- Navigate to “Templates and Presettings” > “Form and Category Collections” > “Your desired form collection”.
- It is assumed that there are contracts defined as form (“Contract”) with a *State* combo box (*Programming Name*: “contractstate”) defining three states.
- Create a new form with “Compound Type” as *Base Class*. The compound type will be assigned to a property on the second form. Define two input fields:
  - State (*Type*: “String”, *Programming Name*: “ctstate”)
  - Amount (*Type*: “Integer”, *Programming Name*: “ctamount”)
- Create a new form and define two item lists:
  - Contracts (*Type*: “Object”, *Default Form for Objects in Field*: “Contract”, *Programming Name*: “contracts”)

- Chart (*Type*: "Compound Type", *Compound Type User Form*: the previously defined compound type, *Compound Type Display Mode*: "Chart", *Expression for Calculating the Control Options*: `dictionary({ chart: { type: "pie" } });`)
  Expression for Computing the Value:

```
Object[] @contracts = cooobj.contracts;
value = [
  {
    ctstate = "In Progress",
    ctamount = count(@contracts[contractstate == 1])
  },
  {
    ctstate = "Review",
    ctamount = count(@contracts[contractstate == 2])
  },
  {
    ctstate = "Approved",
    ctamount = count(@contracts[contractstate == 3])
  }
];
```

  The values for the chart are taken from the *Contracts* field using the expression for computing the value. The chart should be a pie chart.



6.2 Using Fabasoft app.ducx Expressions

To apply some logic or dynamic behavior to the fields, Fabasoft app.ducx expressions can be used.

When you hover over a user-defined field in the form editor, the "Edit" button is displayed. Click it to edit the field, then click the "Advanced" or "Display" tab. Here you can enter the desired Fabasoft app.ducx expression logic.

The following expressions can be defined in the context of forms:

Form: Prepare Commit Configuration

Defines an app.ducx expression that is called when the object is prepared for the commit of the transaction (see also FSCUSERFORMS@1.1001:pccgexpr).

Form Page: Compute Visibility of Page

Defines a Fabasoft app.ducx expression that computes the visibility of a form page (see also COOATTREDIT@1.1:formpagevisibleexpr). Click the "Edit" button of the form page itself, then define the desired expression.

Example

```
// the page is not shown in explore mode (widget in the content area)
::context != #COODESK@1.1:ExploreObject;
```

Compute Visibility of Field

Defines a Fabasoft app.ducx expression that computes the visibility of a property (see also COOSYSTEM@1.1:attrvisibleexpr).

Note: Specified field widths are always considered. This can result in gaps if several fields are defined in a line and one or more of them are hidden by visible expressions.

Example

```
// the field is only shown if it contains a value
// may be useful if it is a programmatically set read-only field
cooobj.informationtype;
```

Compute Value of Field

Defines a Fabasoft app.ducx expression that computes the value of a property (see also COOSYSTEM@1.1:attrvalueexpr). A property with a Fabasoft app.ducx expression for computing its value is always displayed as read-only in the user interface.

Example

```
// FSCUSERFORMS@1.1001:GetFieldValue is used to read the the index of
// the combo box with programming name "informationtype"
// the index of the current combo box is set to 2 or 1 depending on the
// value of the combo box "informationtype"
switch(cooobj.informationtype) {
  case 1: // user manual
  case 2: // service manual
    @availability = 2; // external
    break;
  case 3: // specification
    @availability = 1; // internal
    break;
}
@availability; // return value
```

Handle Initialization of Field

Defines a Fabasoft app.ducx expression that computes the initialization value of a property (see also COOSYSTEM@1.1:attrinitexpr).

Example

```
// object property: when creating an object the initial value will be an
// object with ID COO.1.506.3.4876
COO.1.506.3.4876
```

Compute Changeability of Field

Defines a Fabasoft app.ducx expression that computes the changeability of a property (see also COOSYSTEM@1.1:attrchangeableexpr).

Example

```
// the field can only be edited when the object is created
cootx.IsCreated(cooobj);
```

Filter Values

Defines a Fabasoft app.ducx expression that specifies the selectable values of a property (see also COOSYSTEM@1.1:attrfilterexpr).

Example

```
// it is assumed that there is an object list with programming
// name "contractdocs"
// the following filter is assigned to an object property "maindoc"
// only objects contained in "contractdocs" with category
// COO.1.506.3.4201 are provided for selection in "maindoc"
OBJECTLIST(contractdocs[COOTC@1.1001:objcategory == COO.1.506.3.4201]);
```

Filter Values Within Search

Defines a Fabasoft app.ducx expression that specifies the selectable values of a property in context of a search (see also COOSYSTEM@1.1:attrsearchfilterexpr).

Handle Changes of Values

Defines a Fabasoft app.ducx expression that handles the change of a value of a property in the user interface (see also COOSYSTEM@1.1:attruichangeexpr).

Example

```
// the integer property "days" is set, if the date properties change and
// the conditions are met (must contain values, enddate >= startdate)
// the following expression is assigned to the date properties
// "startdate" and "enddate"
if (cooobj.startdate && cooobj.enddate) {
  if (cooobj.enddate >= cooobj.startdate) {
    cooobj.days = (cooobj.enddate - cooobj.startdate) / 86400;
  }
}
```

Compute Whether the Property Must Be Defined

Defines a Fabasoft app.ducx expression that computes whether the property must contain a value (see also COOSYSTEM@1.1:attrmustbedefexpr).

Example

```
// it is assumed that there is an integer property "importance"
// the following expression is assigned to a property "approvedby"
// if the importance value exceeds 10000 "approvedby" is mandatory
cooobj.importance > 10000;
```

Validate Value

Defines a Fabasoft app.ducx expression that validates a changed value of a property in the user interface (see also COOSYSTEM@1.1:attrvalidateexpr).

Example

```
// if the value of the float property is lower than 0, an error is shown
::value >= 0;
```

Handle Reading of Field

Defines a Fabasoft app.ducx expression that is called before the value of the property is read (see also FSCUSERFORMS@1.1001:attrgetexpr).

Handle Saving of Field

Defines a Fabasoft app.ducx expression that is called before saving the value of the property when committing a transaction (see also FSCUSERFORMS@1.1001:attrsetexpr).

Example

```
// it is assumed that there is an object property "maindoc"
// the following expression is assigned to a property "approvedby"
// the subject of the document in "maindoc" will be set to the value of
// "approvedby"
cooobj.maindoc.ObjectLock(true, true);
cooobj.maindoc.objsubject = value;
```

Read Display String

Defines an app.ducx expression that is called when a string representation of the value of the property is needed (see also FSCUSERFORMS@1.1001:attrgetdispexpr).

Handle Copying of Field

Defines an app.ducx expression that is called when the property is copied (see also FSCUSERFORMS@1.1001:attrcopyexpr). See also the execution order at the end of this chapter.

Handle Construction of Field

Defines an app.ducx expression that is called when the property is created (see also FSCUSERFORMS@1.1001:attrctorexpr). If you just want to set the initial value of the property use *Handle Initialization of Field*. *Handle Construction of Field* can be used to run more general code.

Note: If you use the construction of the field to define an initial value, you may also want to define `COOSYSTEM@1.1:NoOperation` as copy action to keep the initial value for the copied object (see also the execution order at the end of this chapter).

Handle Destruction of Field

Defines an app.ducx expression that is called when the property is deleted (see also FSCUSERFORMS@1.1001:attrdtorexpr).

Compute Control Styles

Defines an app.ducx expression that computes the control styles used to display the property (see also COOSYSTEM@1.1:attrcontrolstyleexpr).

Example

```
// disables the create and search button for an object property
[COOATTREDIT@1.1:ControlStyle(CTRLSTYLE_DISABLECREATE),
 COOATTREDIT@1.1:ControlStyle(CTRLSTYLE_DISABLESEARCH)]
```

Compute Control Options

Defines an app.ducx expression that computes the control options used to display the property (see also COOSYSTEM@1.1:attrcontroloptionsexpr).

Example

```
// the control options are applied to a multiline text property of
// type HTML only the basic styles toolbar is displayed and the height
// is fixed to 10 em
dictionary dict = {
  toolbarGroups: "basicstyles",
  height: "10em"
};
return dict;
```

Execution Order

If you define the following expressions, the execution order will be as follows (e.g. when creating an object based on the form):

1. Handle Construction of Field
2. Handle Copying of Field
3. Handle Initialization of Field
4. Prepare Commit Configuration
5. Handle Saving of Field

Note: The expressions can also influence the draft and release templates of the form, because also in this case construction, initialization and copy expressions are evaluated.

7 Categories

Categories can be assigned to objects (“General” tab) and thus influence the behavior of the objects.

Note:

- An inbox can be used to assign categories that are automatically based on rules.
- The fields of user-defined forms are persisted in a category, which can also define the following settings.

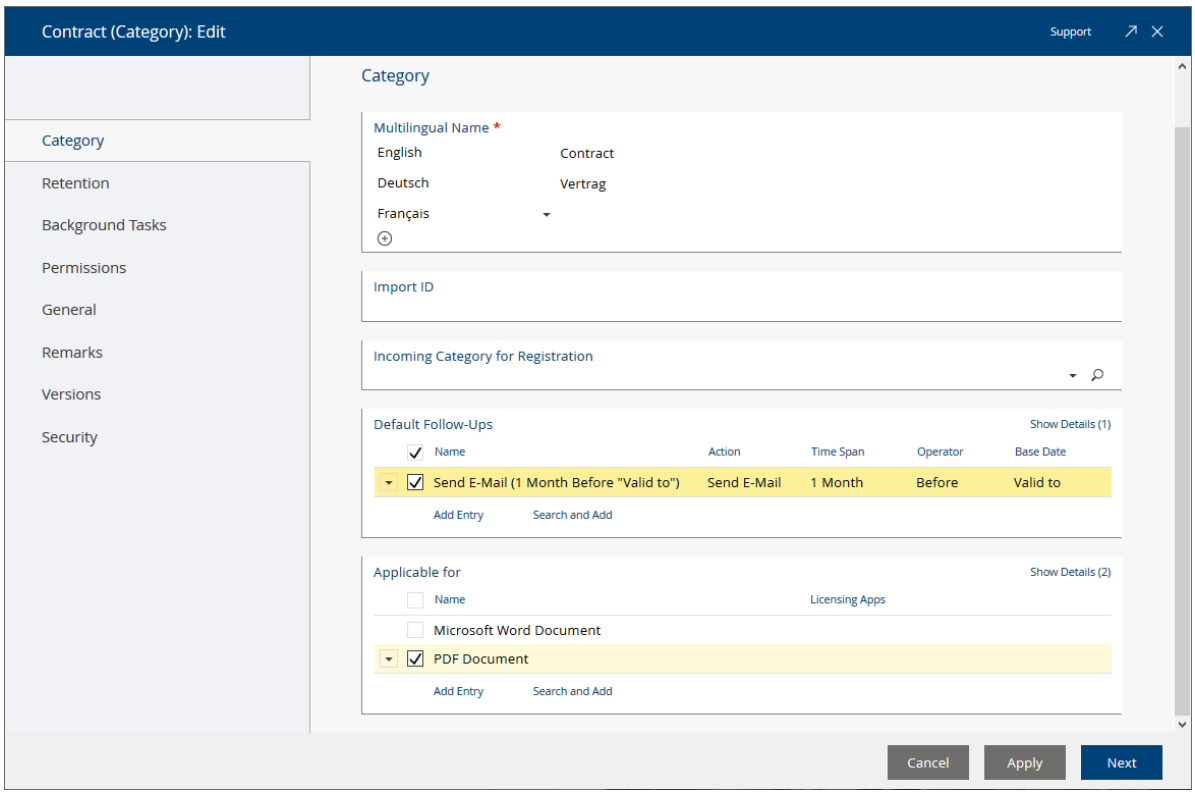
Useful for the Following Tasks

- Defining the registration behavior

- Defining default follow-ups
- Defining the retention worthiness
- Defining background tasks
- Granting additional permissions

Essential information about categories can be found in the user help:

https://help.cloud.fabasoft.com/index.php?topic=doc/User-Help-Fabasoft-Cloud-eng/advanced-use-cases.htm#categories



7.1 Defining the Registration Behavior

By default, the “Register as” context menu command provides all available registration targets. If the registration target is well-known, it can be restricted to improve usability.

Objective
Images that are placed in an inbox should only be registered as digital assets.

To achieve the objective, proceed as follows:

- Edit the properties of the category and select the entry “Incoming Category for Digital Assets” in the *Incoming Category for Registration* field.
- Create an inbox as described in chapter 12 “Inboxes” and define a rule that assigns the category to pictures.

7.2 Defining Default Follow-Ups

If tasks like reviewing documents must be carried out on a regular basis, it is convenient to get a reminder. Categories can provide default follow-ups.

Objective
Confidentiality agreements must be reviewed one month before the validity ends. The responsible users should receive an e-mail reminder.

To achieve the objective, proceed as follows:

- Edit the properties of the category and define a follow-up in the *Default Follow-Ups* field.
- Select the “Send E-Mail” action, then define recipients and the message. You can schedule the follow-up one month before the *Valid to* base date. In addition, the follow-up can be rescheduled when the base date changes. If the user changes the validity after the review, the follow-up will be scheduled again one month before the new validity period ends.

7.3 Defining the Applicability

Not every category makes sense for every object class. Thus, the object classes for which the categories are allowed can be restricted. In addition, if you want to use special base dates for follow-ups or retention periods, a restriction to the object classes that provide the desired properties is needed.

Objective
The category “Contract” only makes sense for Microsoft Word documents and PDF documents. The category should not be assignable to objects of other object classes.

To achieve the objective, proceed as follows:

- Edit the properties of the category and select “Microsoft Word Document” and “PDF Document” in the *Applicable for* field.

7.4 Defining the Retention Worthiness

Compliance rules may enforce that objects must not be deleted for a defined time period.

Objective
Contracts must not be deleted for seven years.

To achieve the objective, proceed as follows:

- Edit the properties of the category and enable the *Retention Worthy* field (“Retention” tab). Assign “7 Years” to the retention period and “Created on/at” as base date.
- Define a background task (“Create Background Task” button) that runs immediately after “Created on/at”. In this case recalculation is not required because the creation date will not change.

Note: The actual retention period can be found on the “Retention” tab of the object (if a retention period is defined).

7.5 Defining Background Tasks

Background tasks allow the execution of an action at a definable point in time.

Objective
A process should be started whenever the *Valid From* date is changed.

To achieve the objective, proceed as follows:

- Edit the properties of the category and create a new background task (“Background Tasks” tab).
- As action, select “Start Process”, and schedule it immediately after the “Valid From”.
- Enable the recalculation when the base date changes, and enable the repeated execution.

7.6 Granting Additional Permissions

In general, the defined team can access the Teamroom and its contents. Access to individual objects can also be granted via a category. The *Access Protection* type of the Teamroom has to be “Extended by Category” so that the categories are considered.

Objective
The “Construction” department should get access to some documents in the “Development” Teamroom.

To achieve the objective, proceed as follows:

- Edit the properties of the category, and define the change/read access permissions for the “Construction” department.
- Set the access protection for the “Development” Teamroom to “Extended by Category”.

8 Processes

BPMN process diagrams are used to model business processes which can be directly executed. Aside from predefined BPMN processes, ad hoc processes can be started whenever they are needed.

Useful for the Following Tasks
• Modeling processes in a straight-forward and graphical way • Tailoring processes to the requirements of your organization • Involving several departments in the processing of business objects • Traceable approving and releasing of documents • Sequential or parallel execution of activities • Using ad-hoc processes

Essential information about processes can be found in the user help:

https://help.cloud.fabasoft.com/index.php?topic=doc/User-Help-Fabasoft-Cloud-eng/workflow.htm

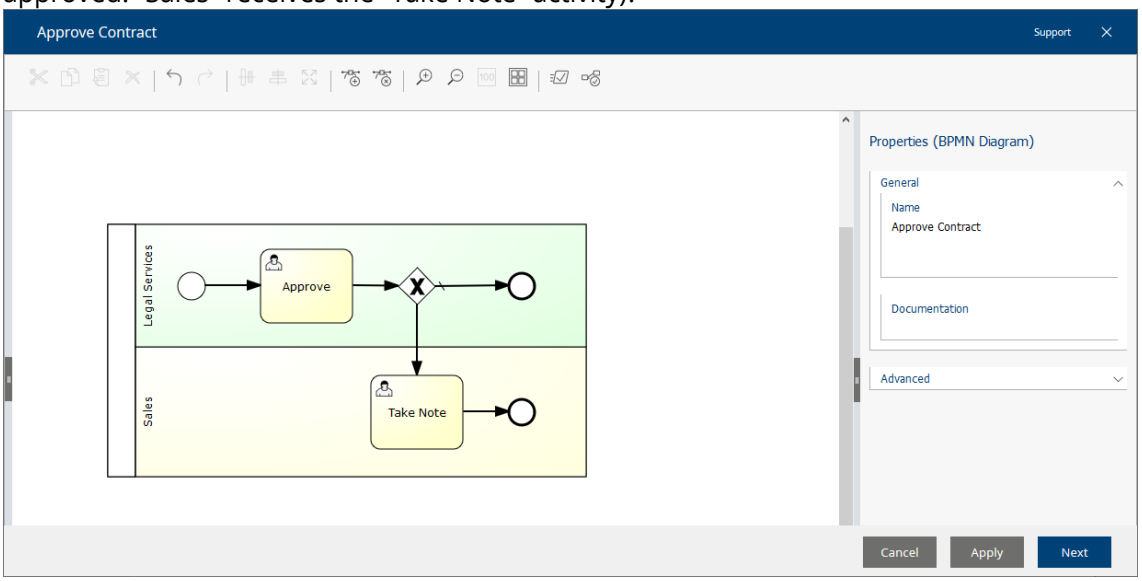
8.1 Scenario

Objective

The “Legal Services” department may only release contracts that are approved by the head of department. The “Sales” department should be informed about each approved contract.

To achieve the objective, proceed as follows:

- It is assumed that the contract is a Word document extended by a user-defined form that provides a *Responsible User* field (*Standard Objects*, *Allowed Standard Objects*: “User”).
- In addition, an organizational structure has to be available (your organization > *Membership* > *Organizational Structure*) that defines the “Legal Services” and “Sales” departments (members and a head).
- Navigate to “Templates and Presettings” > “Process Collections” > “your desired process collection”.
- Define a BPMN process.
- Add a “Pool” and set the *Is Executable* option to “Yes” (properties of the pool).
- Restrict *Applicable for* (properties of the pool) to the category of the user-defined form. This way, the properties of the form are available for modeling the process.
- Define one lane for “Legal Services”. For *Abstract Participant*, select “Role by Property of the Object”. For *Property*, select the *Responsible User* property as defined by the form. As *Position*, select “Head”. This way, the “Legal Services” department will be evaluated because the responsible user is a member of this department. The “Head” is the head of the “Legal Services” department who will receive the activity.
- Define one lane for “Sales”. As *Organizational Unit*, select the “Sales” department. This way all members of the “Sales” department receive the “Take Note” activity.
- Add a “Start Event”, then a task, then an exclusive gateway (not approved: the process ends; approved: “Sales” receives the “Take Note” activity).



- Select the “Approve” activity as the first task. The participant is taken from the lane.
- Exclusive gateway:
  - “Not Approved” sequence flow
    Define the *Condition Type* “Default Flow”.
  - “Approved” sequence flow
    Open the condition editor and select “Approve” in the *Last Signature Type* field (“Signatures” tab).
- Select “Take Note” activity for the second task (“Approved” sequence flow). The participant is taken from the lane.

Processes can be started the following way:

- Manually, by executing the “Tools” > “Start New Process” context menu command on the corresponding object.
- Categories may define background tasks with a “Start Process” action.
- An inbox rule can also be used to start a process.

8.2 Using Fabasoft app.ducx Expressions

Following expressions can be defined in the context of processes.

Note: Using the “Show Overview of app.ducx Expressions” context menu command, you get an overview of all app.ducx expressions defined in the process and, if applicable, in the subprocesses. This facilitates troubleshooting in particular.

8.2.1 Process Elements

The following expressions are available in the context of process elements.

Expression When Completing the Activity

Defines a Fabasoft app.ducx expression that is executed when the activity is completed (see also COOWF@1.1:wfcompleteexpression).

To define the expression, double-click a task and switch to the “Extended” tab.

Example

```
// the object on which the process runs and the current activity are
// available in the local scope
// the processing state of the object will be set
object.ObjectLock(true, true);
object.FSCFOLIO@1.1001:bostate = #FSCFOLIO@1.1001:StateToVerify;


// another process will be started based on a
// BPMN process diagram (e.g. COO.1.506.2.3642)
// the process instance will be stored as global process parameter
COOWF@1.1:ProcessInstance @instance =
  object.COOWF@1.1:StartProcessDiagram(COO.1.506.2.3642);
process.COOWF@1.1:SetProcessParameter("proc1", @instance);


// the previously started process can be conditionally terminated in a
// following activity
if (object.FSCFOLIO@1.1001:bostate == #FSCFOLIO@1.1001:StateDone) {
  COOWF@1.1:ProcessInstance @instance =
```

```
    process.COOWF@1.1:GetProcessParameter("proc1");
  @instance.COOWF@1.1:SetProcessTerminated();
}
```

Execute Expression in Background

The *Execute Expression in Background* activity can be used to execute an expression in background instead of providing the activity to the user (see also FSCDIAGRAMEDITOR@1.1001:wfbackgroundexpression). The process is not continued until the background task has been executed.

To define the expression, double-click a task and select “Execute Expression in Background” as *Activity*.

Example

```
// sends an e-mail to the responsible user
import FSCFOLIO@1.1001;
import LASC_1_1384RELEASEFORMS@1.506;
coouser.SendBackgroundSecure(object.responsibleuser, null, null, "My Subject",
"My Body Text");
```

Loop Condition

Defines a Fabasoft app.ducx expression that is used to evaluate the loop condition of an activity (see also COOWF@1.1:wfwexpression).

To define the expression, select a task and define a *Loop Type*. Edit the *Loop Condition* of the task. You can either directly enter a Fabasoft app.ducx expression, or use the search-like editor.

Example

```
// the loop continues as long as the object has the wrong state
object.FSCFOLIO@1.1001:bostate == #FSCFOLIO@1.1001:StateToVerify;
```

Path Condition

Defines a Fabasoft app.ducx expression that is used to evaluate whether a gateway path should be followed (see also COOWF@1.1:pathcondition).

To define the expression, select a sequence flow and edit the *Condition* property of the sequence flow. You can either directly enter a Fabasoft app.ducx expression, or use the search-like editor.

Example

```
// the sequence flow is followed if the process parameter "isprepared"
// is true
process.COOWF@1.1:GetProcessParameter("isprepared") == true;
```

Intermediate Conditional Event

Defines a Fabasoft app.ducx expression that stops further processing until the condition is fulfilled (see also FSCDIAGRAMEDITOR@1.1001:eventpropscondition).

To define the expression, select an intermediate conditional event, then edit the *Condition* property of the intermediate conditional event.

Example

```
// the process stops until the property has a value
object.HasAttributeValue(cootx, #LASC_1_1384RELEASEFORMS@1.506:responsibleuser);
```

8.2.2 Process Diagram

The following expressions are available in the context of the process diagram.

Expression for Determining the Visibility

Defines a Fabasoft app.ducx expression that determines whether the process is offered for selection when a process is started (see also COOWF@1.1:wfvisibleexpression).

To define the expression, select the pool and edit the *Applicable for* property of the pool.

Example

```
// the process is only visible for starting if the "Responsible User"
// property (programming name: "responsibleuser") has a value
object.HasAttributeValue(cootx, #LASC_1_1384RELEASEFORMS@1.506:responsibleuser);
```

Expression for Determining the Usability

Defines a Fabasoft app.ducx expression that determines whether the process can be started. This allows, for example, to check preconditions that must be fulfilled before the process can be started (see also COOWF@1.1:wfprecondexpression). If the preconditions are not fulfilled, an error message is shown.

To define the expression, select the pool and edit the *Applicable for* property of the pool.

Example

```
// a customized error is shown if the process is started on an object
// that is still valid
// to show the generic error message just enter an expression that
// returns a Boolean value
import COODESK@1.1;
if (object.objvalidto >= coonow){
  coouser.RaiseError(#ErrorGeneric, "The object is still valid.");
}
else {
  true;
}
```

Expression for Initializations

Defines a Fabasoft app.ducx expression that allows defining common initializations and global process parameters (see also COOWF@1.1:wfinitializationexpression).

To define the expression, select the pool and edit the *Initializations* property of the pool.

Example

```
// initially defined process parameters can be read in other process
// expressions using: process.GetProcessParameter("isprepared");
process.SetProcessParameter("isprepared", true);
process.SetProcessParameter("anotherkey", "another value");
```

9 Templates and Text Modules

In the “Templates and Presettings” area, you can create template and text module collections, manage templates and text modules therein, and make them available to users.

Useful for Following Tasks

- Providing templates for individual departments.
- Using metadata as fields in Microsoft Word documents.
- Defining personal templates.
- Using text module placeholders in Microsoft Word templates (will be replaced when an instance is created).
- Using text modules (predefined text) in Microsoft Word documents.

Essential information about templates and text modules can be found in the user help:

https://help.cloud.fabasoft.com/index.php?topic=doc/User-Help-Fabasoft-Cloud-eng/customizing.htm

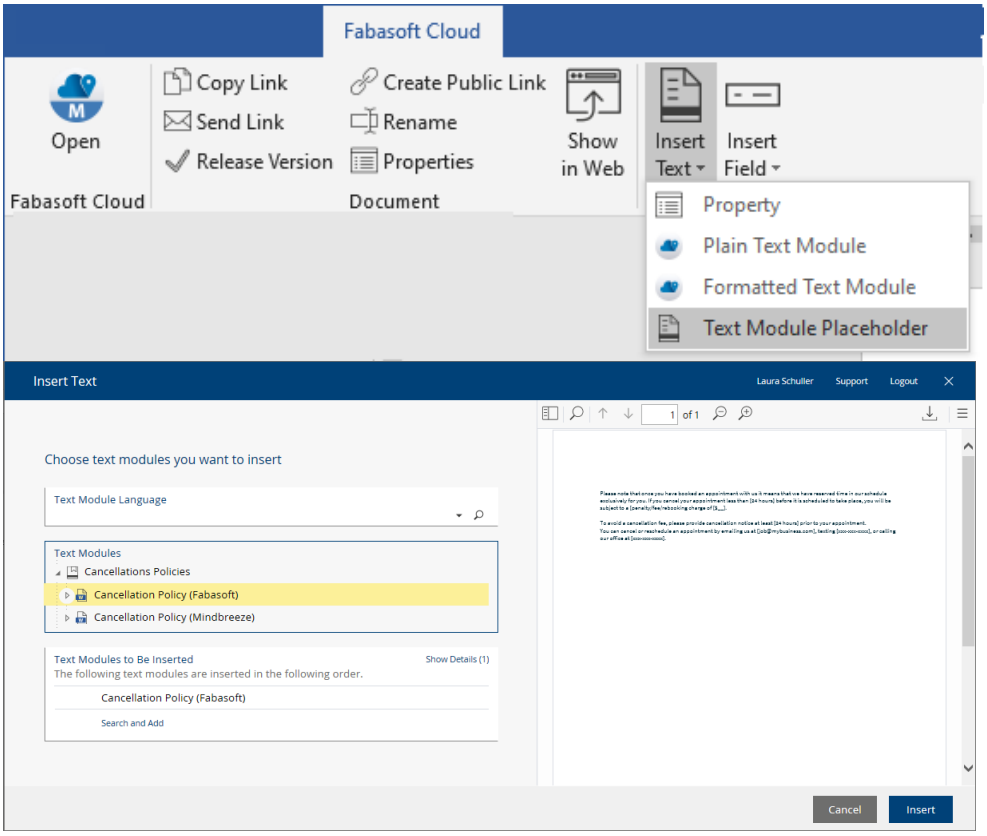
9.1 Scenario

Objective

For each contract type a Word template must exist. The current version of standard contract text blocks must be inserted when a contract is created based on the template.

To achieve the objective, proceed as follows:

- Navigate to “Templates and Presettings”.
- In a template collection, create a Microsoft Word template for each contract type.
- In a text module collection, create a text module for each standard text block.
- Edit the Word document, insert the contract text and insert text module placeholders for standard text blocks (“Fabasoft Cloud” tab > “Insert Text” > “Text Module Placeholder”).



- When a contract based on the template is created, the text module placeholders are replaced.

The usage of the template can be restricted to improve usability and to define available Word fields (“Template” tab).

- *Category*
  The category is copied to the object created from the template.
  Note: If applicable, lists can programmatically allow objects only with certain categories. In this way, the category can also restrict where the template can be used.
- *Contained in*
  Defines the folder types in which the template can be used. This can either be done using the object class or the category of the folder. If object classes and categories are specified together, the folder must have a defined object class and a defined category.
- *Files*
  Defines the type of files in which the template can be used. You can do this using either the object class or the category of the file. If object classes and categories are specified together, the file must have a defined object class and a defined category.

9.2 Using Fabasoft app.ducx Expressions

The following expressions can be defined in the context of text modules.

Expression for Further Restrictions of Usability

Defines a Fabasoft app.ducx expression that may further restrict the use of the text module (see also COOAR@1.1:modulevisibleexpr).

The expression can be found on the “Usage” tab of the text module.

Example

```
// can only be used in objects that contain the string "(draft)"
// in the name
lower(objname) contains "(draft)"
```

10 Display Settings

Predefined display settings can be provided to users who need special views on lists.

Useful for Following Tasks

- Providing an overview of object metadata

Essential information about display settings can be found in the user help:

https://help.cloud.fabasoft.com/index.php?topic=doc/User-Help-Fabasoft-Cloud-eng/customizing.htm#defining-presettings

10.1 Scenario

Objective

The “Controlling” department needs an overview of contract metadata.

To achieve the objective, proceed as follows:

- Navigate to “Templates and Presettings” > “Presetting Collections” > “desired presetting collection”.
- Create a display setting.
- Define which columns should be displayed, and how (sorting, grouping, width).

11 Search Forms

Predefined search forms can be provided to users who need an overview of currently existing objects based on search criteria.

Useful for Following Tasks

- Providing an overview of currently existing objects based on search criteria.

Essential information about search forms can be found in the user help:

https://help.cloud.fabasoft.com/index.php?topic=doc/User-Help-Fabasoft-Cloud-eng/customizing.htm#defining-presettings

11.1 Scenario

Objective

The “Legal Services” department needs all contracts of a certain year per business unit.

To achieve the objective, proceed as follows:

- Navigate to “Templates and Presettings” > “Presetting Collections” > “desired presetting collection”.
- Create a search form.
- Define the search criteria (object class and values of properties).

12 Inboxes

In an inbox, rules for the processing of incoming objects can be defined. A rule consists of conditions and actions. For example, network drives (WebDAV) can be used to store external documents in the inbox.

Useful for Following Tasks
• Importing an organizational structure • Extracting e-mail attachments • Defining property values • Assigning categories • Classifying with Mindbreeze InSpire • Starting processes

Essential information about inboxes can be found in the user help:

https://help.cloud.fabasoft.com/index.php?topic=doc/User-Help-Fabasoft-Cloud-eng/advanced-use-cases.htm#inbox

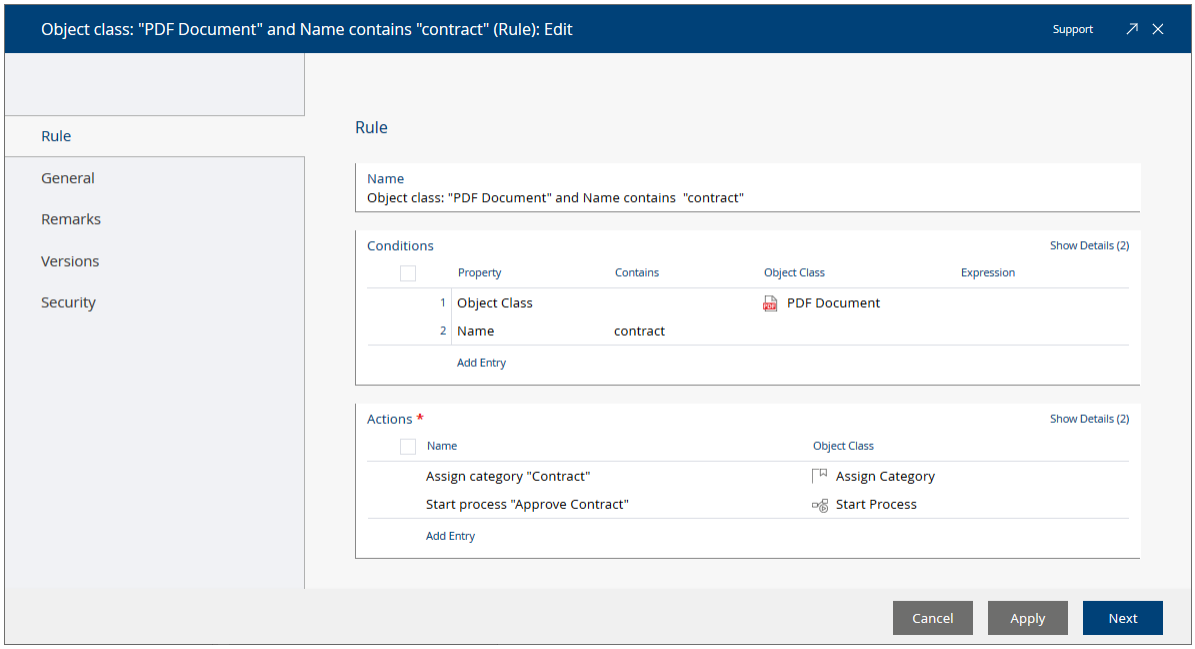
12.1 Scenario

Objective
If the incoming object is a PDF document and the name of the object contains the word “contract”, then the “Contract” category should be assigned. In addition, a process for approving should be started.

To achieve the objective, proceed as follows:

- Create an inbox (in the personal folder or in a Teamroom).
- Define a rule:
  - As the first condition, select “Object Class” as property and “PDF Document” as value.
  - As the second condition, select “Name” as property and enter “contract” as value.
  - As the first action, use “Assign Category” and select the “Contract” category.
  - As the second action, use “Start Process” and select the desired BPMN process.

Note: All conditions must be met for the rule to be executed. Do not specify a condition if the rule should always be executed. All actions of a rule are executed, but only the first applicable rule is executed.



12.2 Using Fabasoft app.ducx Expressions

The following expressions can be defined in the context of inboxes.

Condition as Expression

Defines a Fabasoft app.ducx expression that is used as rule condition (see also FSCTEAMROOM@1.1001:rlcexpression).

Example

```
// the rule action is executed if the name of the incoming object
// contains "contract" (case-insensitive)
// "objname" can be read as "this.objname" whereby "this" is
// the incoming object
lower(objname) contains "contract"
```

Execute User Defined Expression

The “Execute User Defined Expression” action can be used to execute an expression on the incoming object (see also FSCTEAMROOM@1.1001:eeaexpression).

Example

```
// sets the subject of the incoming object
objsubject = "Subject of " + objname;


// the applied rule is available in the global scope
// the incoming object will be removed from the inbox list
// the incoming object is added to a Teamroom with
// Cloud ID "COO.1.506.3.4282"
@inbox = ::rule.FSCTEAMROOM@1.1001:GetObjectRoom();
@inbox.ObjectLock(true,true);
@inbox.FSCTEAMROOM@1.1001:ibrchildren -= this;
@tr = COO.1.506.3.4130;
@tr.ObjectLock(true,true);
@tr.FSCTEAMROOM@1.1001:trchildren += this;
```

13 Comprehensive Example: Handling Documents and Collaborating

This comprehensive example summarizes the most important customization options within one extensive scenario to provide an insight into how to solve an application-specific problem.

Objective

Starting Situation

A plant construction and engineering company is divided into following scenario-relevant departments:

- Development
- Construction
- Sales

The main objective is to provide an efficient way to handle technical documents and to simplify the collaboration within the organization and with suppliers/customers.

- The “Development” department depends on specification documents from suppliers, and authors custom specification documents.
- The “Construction” department depends on the documents of the “Development” department.
- The “Sales” department provides contracts for customers.

Specification Documents

- A product file summarizes all specification documents of a product.
- Standard product files are provided as templates.
- Standard specification documents are provided as templates.
- The product file metadata is used in the specification document templates as fields.
- The detail view with defined columns for the most important product metadata is used to provide a quick overview.
- Processes are used to inform other departments of approved specification documents.

Collaboration With Customers

- A contract is modelled as a Word document with additional metadata provided by a form.
- Standard contracts are provided as templates.
- Text modules are used to provide standard text blocks that can be added to individual contracts.
- The contracts are published as PDF documents with QR codes and sent to the customer to be signed.
- Contracts received as e-mail attachments can be automatically imported to the Fabasoft Cloud using the Microsoft Outlook integration. An inbox is used to extract the attachments and automatically replace the unsigned contracts with the signed ones.

13.1 Common

Model-based customizations are only accessible to users who are defined as team members in the customizing rooms. For usability reasons, do only provide customizations to the users that need them. In addition, the customizations have to be released for usage.

13.2 Specification Documents

The following model-based customizations are carried out to manage specification documents.

Task 1

- A product file summarizes all specification documents of a product.
- Standard product files are provided as templates.

Product File Form

1. Navigate to “Templates and Presettings” > “Form and Category Collections” > “desired form collection”.
2. Create a form with two pages. The “Product” page should not be visible in explore mode while the “Documents” page should only be visible in explore mode.
   - *Base Class*: “Container With User Data”
   - *Use as File*: “Yes”
   - “Product” page
     Expression for Computing the Visibility: `::context != #COODESK@1.1:ExploreObject;`
     - *Product ID*
       “Input Field”, *Type*: “String”, *Programming Name*: “prodid”
     - *Product Description*
       “Input Field”, *Type*: “String”, *Programming Name*: “proddesc”
     - *Responsible User*
       “Input Field”, *Type*: “Object”, “Standard Objects“, *Allowed Standard Objects*: “User”, *Programming Name*: “responsible user”
     - Any additional fields that might be needed.
   - “Documents” page
     Expression for Computing the Visibility: `::context == #COODESK@1.1:ExploreObject;`
     - *Documents*
       “Input Field”, *Type*: “Object”, “Commonly Usable Objects”, *Allowed Commonly Usable Objects*: “Object With Object List” and “Document”, *Programming Name*: “proddocuments”

Product File Template

1. Navigate to “Templates and Presettings” > “Template Collections” > “desired template collection”.
2. Create a template (based on the “Product” form that you created before).
3. Create a template category named “Products” and assign it to the template (“Template” tab > *Template Categories*), so that the template is displayed in a separate group in the creation dialog.

4. You can further edit the template in order to create a predefined folder hierarchy in the *Documents* field, for instance.

Testing

1. Navigate into a Teamroom and create a new object (“Add Entry” > “New”). The product file is provided in the “Products” group.
2. A specification document can be added to the product file, for instance.

Task 2

- Standard specification documents are provided as templates.
- The product metadata is used in the specification document templates as fields.

Specification Document Template

1. Navigate to “Templates and Presettings” > “Template Collections” > “desired template collection”.
2. Create a template (based on a “Microsoft Word Document”).
3. Create a template category named “Specifications” and assign it to the template (“Template” tab > *Template Categories*), so that the template is displayed in a separate group in the creation dialog.
4. For the product file metadata to be used as fields in the Word document, assign the product file category to the template (“Template” tab > *Files*).
5. Edit the Word document and add the standard text of the specification. Product file metadata can be inserted as needed (e.g. “Fabasoft Cloud” tab > “Insert Field” > “Product” > “Product ID”).

Testing

1. Create a Word document in a product file.
2. Edit the document and verify that the standard text is present, and that the fields are replaced with product file metadata accordingly.

Task 3

The detail view with defined columns for the most important product metadata is used to provide a fast overview.

Display Settings

1. Navigate to “Templates and Presettings” > “Presetting Collections” > “desired presetting collection”.
2. Create a “Display Setting” and define the desired columns.

Testing

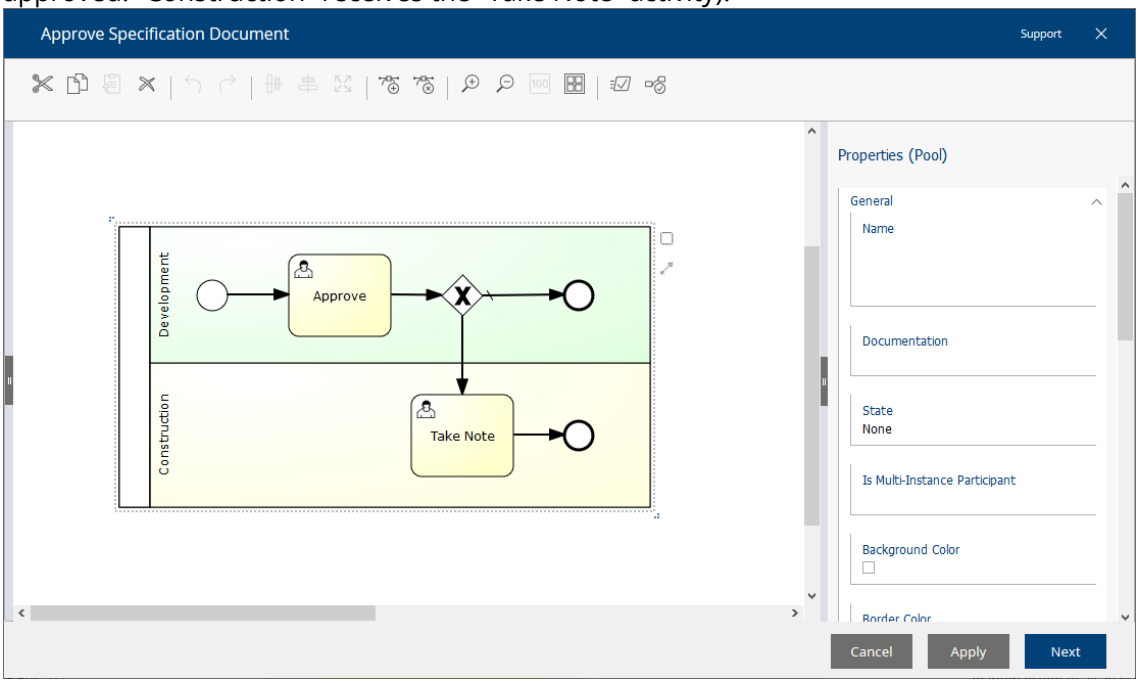
1. Navigate into a Teamroom that contains product files.
2. Click on the entries count on the top right and select the previously defined display setting.

Task 4

Processes are used to inform other departments of approved specification documents.

BPMN Process

1. Navigate to “Templates and Presettings” > “Process Collections” > “desired process collection”.
2. Create a BPMN process.
3. Add a “Pool” and set *Is Executable* to “Yes” (properties of the pool).
4. Edit *Applicable for* in the properties of the pool. Define the category of the user-defined form “Product” in the *Object Class/Category of the File* field. This way, the properties of the form are available in order to model the process.
5. Define one lane for “Development”. As *Abstract Participant* select “Role by Property of the File of the Object”. For *Property,* select the *Responsible User* property as defined by the product file form. For *Position,* select “Head”. This way, the “Development” department will be evaluated because the responsible user is a member of this department. “Head” is defined as the head of the “Development” department who will receive the activity.
6. Define one lane for “Construction”. As *Organizational Unit* select the “Construction” department. This way, all members of the “Construction” department receive the “Take Note” activity.
7. Add a “Start Event”, then a task, then an exclusive gateway (not approved: the process ends; approved: “Construction” receives the “Take Note” activity).



8. Select the “Approve” activity for the first task. The participant is taken from the lane.
   - Exclusive gateway:
     - “Not Approved” sequence flow
       Define the *Condition Type* “Default Flow”.

- "Approved" sequence flow
  Open the condition editor and select "Approve" in the *Last Signature Type* field ("Signatures" tab).

- Select the "Take Note" activity for the second task ("Approved" sequence flow). The participant is taken from the lane.

Testing

1. Navigate to a specification document stored in a product file.
2. Start the defined process ("Tools" > "Start New Process" context menu command).
3. The activities are shown in the worklist of the respective users.

13.3 Collaboration with Customers

Following model-based customizations are carried out to ease the collaboration with customers.

Task 1

- A contract is modelled as a Word document with additional metadata provided by a form.
- Standard contracts are provided as templates.

Contract Form

1. Navigate to "Templates and Presettings" > "Form and Category Collections" > "desired form collection".
2. Create a form:
   - *Base Class*: "Object With User Data"
   - *Suppress Template Creation*: "Yes"
   - Fields
     - *Contracting Party*
       "Input Field", *Type*: "Object", "Standard Objects", *Allowed Standard Objects*: "Contact Person" and "Organization", *Programming Name*: "contractingparty"
     - *Contract Value*
       "Input Field", *Type*: "Currency", *Programming Name*: "contractvalue"
     - Any additionally needed fields.

Contract Word Template

1. Navigate to "Templates and Presettings" > "Template Collections" > "desired template collection".
2. Create a template ("Microsoft Word Document").
3. Assign the *Category (Published)* of the previously defined form to the template ("Template" tab > *Category*).
4. Create a template category named "Contracts" and assign it to the template ("Template" tab > *Template Categories*), so that the template is displayed in a separate group in the creation dialog.
5. Edit the Word document and enter the contract text.

Testing

1. Navigate into a Teamroom and create a new object (“Add Entry” > “New”). The contract is provided in the “Contracts” group.
2. In the properties of the newly created contract, the contracting party can be selected (organizations and contact persons are managed in the “Contact Management”).

Task 2

Text modules are used to provide standard text blocks that can be added to individual contracts.

Text Module

1. Navigate to “Templates and Presettings” > “Text Module Collections” > “desired text module collection”.
2. Create a text module and enter the desired standard text.
3. Create a text module category named “Cancellations Policies” and assign it to the text module (“Template” tab > *Template Categories*), so that the text module is displayed in a separate group in the selection dialog.

Testing

1. Edit the contract and insert a text module at the cursor position (“Fabasoft Cloud” tab > “Insert Text” > “Formatted Text Module”).
2. Select the desired text module.

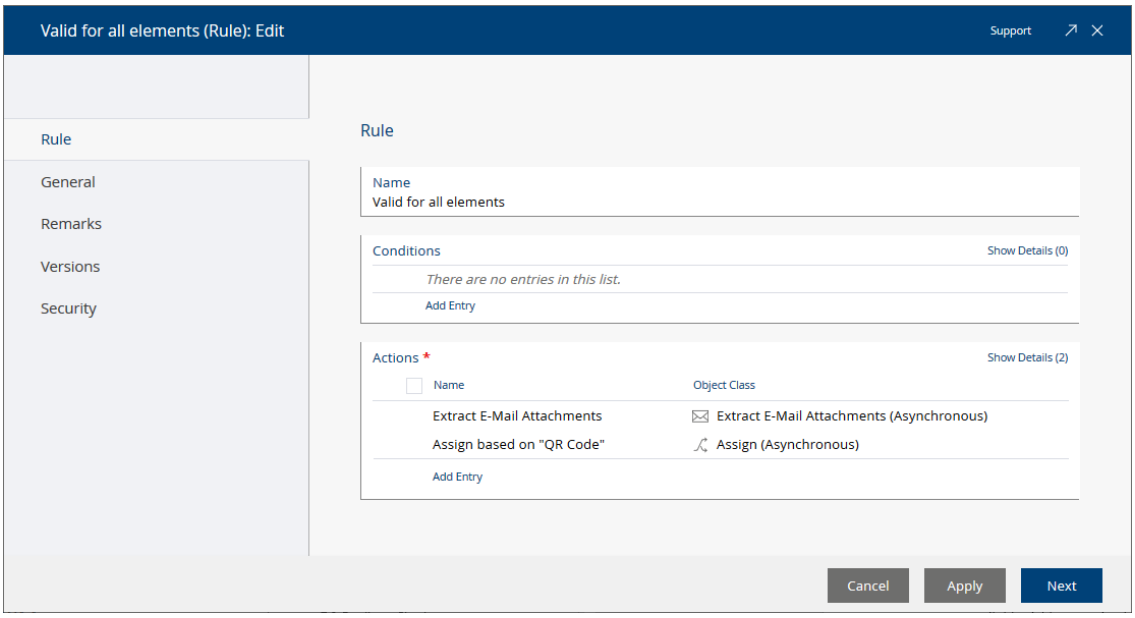
Task 3

- The contracts are published as PDF documents with QR codes and sent to the customer to be signed.
- Contracts received as e-mail attachments can be automatically imported to the Fabasoft Cloud using the Microsoft Outlook integration. An inbox is used to extract the attachments and automatically replace the unsigned contracts with the signed ones.

Inbox

1. Navigate to your personal folder.
2. Create an inbox (background context menu command “New”).

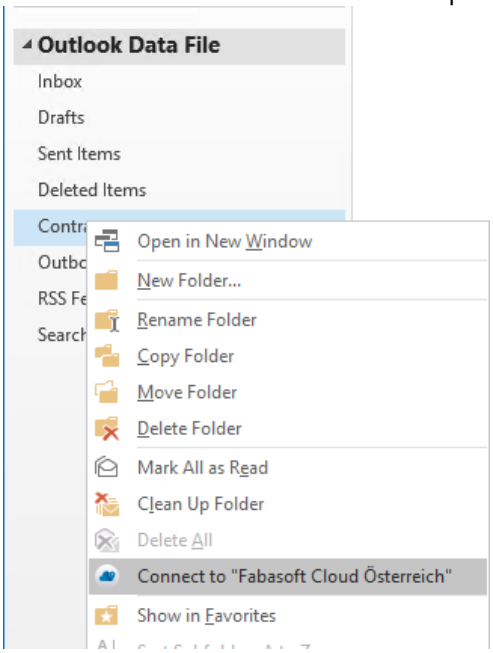
3. Define a rule with two actions.



   - "Extract E-Mail Attachments (Asynchronous)"
     Post-Processing: "Delete E-Mail After Extracting Attachments"
   - "Assign (Asynchronous)"
     Assignment Based on: "QR Code"

Microsoft Outlook Integration

1. In Microsoft Outlook, create a new folder and define an Outlook rule which states that e-mails with attached contracts are moved to this folder (e.g. based on the subject).
2. Execute the "Connect to Fabasoft Cloud" context menu command on the Outlook folder to establish the connection. Select the previously created inbox as target.



Testing

1. Print the contract as PDF with QR code ("Tools" > "Print With QR Code" context menu command).

2. Compose an e-mail in Microsoft Outlook and attach the PDF document with QR code (simulates a customer e-mail).
3. Move the e-mail to the connected Outlook folder.
4. The e-mail will be imported to the inbox. The inbox actions are executed asynchronously, resulting in the Word contract being replaced by the signed PDF contract automatically.

14 Comprehensive Example: Service Process

This example demonstrates how a simple service process can be modeled, using templates, text modules, user-defined forms and BPMN processes – free of media disruption.

Objective

The following service process should be modeled:

Customer

- The customer sends a support request via e-mail.

Support Department

- The e-mail is imported via the “Cloud Mail Import” function.
- An inbox assigns the category “Support Request” to the e-mail and the process “Check Service Request” is initiated.
- The support employee validates the service request in the workflow. For this purpose, the comment function can be used.

Service Department

- The service employee issues a service order for the service request in the workflow.

Field Service

- The field service employee conducts the service order.
- The field service employee completes the service call report in the workflow as predefined in a template and uploads images for documentation purposes.

Support Department

- The support employee closes the service call report in the workflow. For this purpose, the signature function can be used, and thus the service call report can be countersigned.

Service Department

- The service employee closes the service order in the workflow.

In the following chapters, you can find the elements and adjustments necessary for implementing the service process. For the sake of a compact presentation, there is no step-by-step instruction. The necessary basic information is available in the previous chapters.

14.1 Forms

In a form and category collection, create the forms as described in the following chapters and release them for usage. You need to re-release them after an alteration.

- Service Request
  A service request is an e-mail that is categorized as a service request.
- Service Order
  A service order is created from a reviewed service request.
- Service Orders
  Is used as a dashboard to present all service orders in one view.
- Service Call Report
  A service call report is completed by the technician and countersigned by the customer.

14.1.1 Service Request

The "Service Request" form is defined as follows:

Common (Create Dialog)

- *Base Class*: "Object With User Data"
- *Symbol*: "Mail"
- *Suppress Template Creation*: "Yes"

"Service Request" page

- *Expression for Computing the Visibility*: `false;`
  No additional fields are required since the service request is based upon an e-mail that comes with all necessary meta data. Hence, this form page should not be visible.
- Separator
  Add a separator because the form page must contain at least one element.

Common (Properties)

- *Applicable for*: "E-Mail (MIME)", "E-Mail (Microsoft Office Outlook)"

14.1.2 Service Order

The "Service Order" form is defined as follows:

Common (Create Dialog)

- *Base Class*: "Container With User Data"
- *Use as File*: "Yes"
- *Symbol*: "Support Request"

"Check List" page

- *Expression for Computing the Visibility*: `cooobj.FSCFOLIO@1.1001:bostate not in [#FSCFOLIO@1.1001:StateReceived, #FSCFOLIO@1.1001:StatePlanning];`
  The tab should only be visible if this is useful (based on the status)
- *Ball Bearing Lubricated*
  "Check Box", *Programming Name*: "soballbearinglubricated", *Expression for Computing the Changeability*: `cooobj.FSCFOLIO@1.1001:bostate in [#FSCFOLIO@1.1001:StateInProgress];`
- *Remarks*
  "Input Field", *Type*: "String", *Programming Name*: "soremarks"

"Service Order" page

- *Number*
  "Input Field", *Type*: "Numerator", *Numerator Keys*: "Teamroom", *Programming Name*: "sonumerator"
- *Customer*
  "Input Field", *Type*: "Object" ("Standard Objects", "User"), *Programming Name*: "socustomer"
- *Address*
  "Input Field", *Type*: "String", *Programming Name*: "socustomeraddress", *Expression for Computing the Value*:
  ```
  FSCFOLIO@1.1001:Address @address =
  cooobj.socustomer.FSCFOLIO@1.1001:address[0];
  if (@address) {
    @address.FSCFOLIO@1.1001:addrstreet + ", " +
    @address.FSCFOLIO@1.1001:addrzipcode + " " +
    @address.FSCFOLIO@1.1001:addrcity;
  }
  ```
- *Phone*
  "Input Field", *Type*: "String", *Programming Name*: "socustomertelephone" *Expression for Computing the Value*:
  ```
  cooobj.socustomer.FSCFOLIO@1.1001:telephone[0].FSCFOLIO@1.1001:telnumber;
  ```
- *Activity*
  "Input Field", *Type*: "String", *Must Be Defined*: "Yes", *Programming Name*: "soactivity", Expression for Computing the Changeability:
  ```
  cooobj.FSCFOLIO@1.1001:bostate in [#FSCFOLIO@1.1001:StateReceived,
  #FSCFOLIO@1.1001:StatePlanning];
  ```
- *Status*
  "Standard Property", *Standard Property*: "Processing State", *Label*: "Status", *Expression for Computing the Changeability*: `false;`
- *Service Technician*
  "Input Field", *Type*: "Object" ("Standard Objects", "User"), *Must Be Defined*: "Yes", *Programming Name*: "sotechnician", *Expression for Computing the Changeability*:
  ```
  cooobj.FSCFOLIO@1.1001:bostate in [#FSCFOLIO@1.1001:StateReceived,
  #FSCFOLIO@1.1001:StatePlanning];
  ```
- *Service Appointment*
  "Input Field", *Type*: "Date", *Must Be Defined*: "Yes", *Initialize With Current Date*: "Yes", *Programming Name*: "sodate", *Expression for Computing the Changeability*:
  ```
  cooobj.FSCFOLIO@1.1001:bostate in [#FSCFOLIO@1.1001:StateReceived,
  #FSCFOLIO@1.1001:StatePlanning];
  ```

"Documents" page

- *Documents*
  "Item List", *Type*: "Object" ("Commonly Usable Objects", "Document"), *Programming Name*: "sodocuments"

Common (Properties)

- *Applicable for*: "Container With User Data"
- *Available Processing States*: "Received", "Planning", "In Progress", "Done", "Released", "Not Released"
- *Default Processing State*: "Received"

- *Name Build Configuration*
  Note: Before defining the name build, release the form at least once.
  - *Enable Name Build*: "Yes"
  - *Used Properties*: "Name"
  - *Expression for Computing the Name Build*:
    ```
    "SO " + coouser.Format(cooobj.[#sonumerator], "0000") + " - " +
    coouser.Format(cooobj.objcreatedat, "yyyy-mm-dd") + " - " +
    cooobj.socustomer.GetName();
    ```

14.1.3 Service Orders

The "Service Orders" form is defined as follows:

Common (Create Dialog)

- *Base Class*: "Room With User Data"
- *Symbol*: "Tree"

"Service Orders" page

- *Service Orders*
  "Item List", *Type*: "Object" ("Forms", "Service Order"), *Default Form for Objects in Field*: "Service Order", *Programming Name*: "soserviceorders"

Common (Properties)

- *Applicable for*: "Room With User Data"

14.1.4 Service Call Report

The "Service Call Report" form is defined as follows:

Common (Create Dialog)

- *Base Class*: "Document With User Data"
- *Suppress Template Creation*: "Yes"

"Service Call Report" page

- *Expression for Computing the Visibility*: `false;`
  No additional fields are necessary. Hence, this form page should not be visible.
- Separator
  Add a separator because the form page must contain at least one element.

Common (Properties)

- *Applicable for*: "Microsoft Word Document"

14.2 Text Modules

In a text module collection, create the text modules as described in the following chapters.

The text modules are used in the template for the service call report. Depending on the result of technician's check (OK, NOT OK), the text module is inserted in case of a positive or negative result. In order to simplify the example, only the test point "Ball Bearing Lubricated" is defined.

14.2.1 Ball Bearing Lubricated - OK

The static text module (Word) “Ball Bearing Lubricated - OK” contains the text that is to be inserted upon a positive result.

"Usage” tab

- *Current Document*: “Microsoft Word Document”, “Service Call Report”
- *Files*: “Service Order”
- *Expression for Further Restrictions of Usability*:
```
cooobj.FSCFOLIO@1.1001:objfile.[#soballbearinglubricated] == true;
```

14.2.2 Ball Bearing Lubricated - NOT OK

The static text module (Word) “Ball Bearing Lubricated - NOT OK” contains the text that is to be inserted upon a negative result.

"Usage” tab

- *Current Document*: “Microsoft Word Document”, “Service Call Report”
- *Files*: “Service Order”
- *Expression for Further Restrictions of Usability*:
```
cooobj.FSCFOLIO@1.1001:objfile.[#soballbearinglubricated] == false;
```

14.3 Template

In a template collection, create a Microsoft Word document “Service Call Report” with fields, text module placeholders, and image placeholders.

“Template” tab

- *Category*: “Service Call Report”
- *Files*: “Service Order”

Fields (Word Document)

- Customer
- Address
- Service Technician
- Ball Bearing Lubricated

Text Module Placeholders (Word Document)

- Ball Bearing Lubricated - NOT OK
- Ball Bearing Lubricated - OK

Image Placeholders (Word Document)

- Arbitrary image (alternative text, without quotes: “Picture before service call”)
- Arbitrary image (alternative text, without quotes: “Picture after service call”)

COMPANY
ENGINEERING SOLUTIONS

Service Call Report

Wednesday, February 10, 2021

Customer	Customer
Address	Address
Service Technician	Service Technician

Checkpoint	Result	Measure
Ball bearing	☐	Ball bearing lubricated: OK Ball bearing lubricated: NOT OK

Part	Amount

For the contractor	
Place, date	Signature

14.4 Processes

In a process collection, create the processes as described in the following chapters.

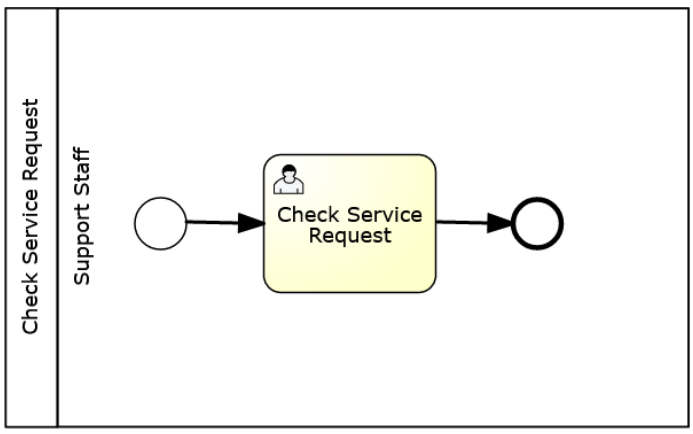
Pre-requisites:

- It is assumed that the following departments exist in the organization: “Support Staff”, “Service Staff”, “Customers” (external organization).
- An instance of the form “Service Orders” must exist (see chapter 14.1.3 “Service Orders”).
- An inbox “Support Inbox” must exist (see chapter 14.5 “Inbox”). At this point, it suffices to have an inbox because the final configuration requires the process “Check Service Request”.

Note: In order to efficiently test the processes, make your user account the sole participant in every step. Only after a successful initial test is it advisable to assign participants for the productive operation of the processes.

14.4.1 Check Service Request

Create the BPMN process “Check Service Request”.



Pool

- *Applicable for*: "Service Request"

Lane

- Support Staff
  - *Process Participant* > *Organizational Unit*: "Support Staff"

Task

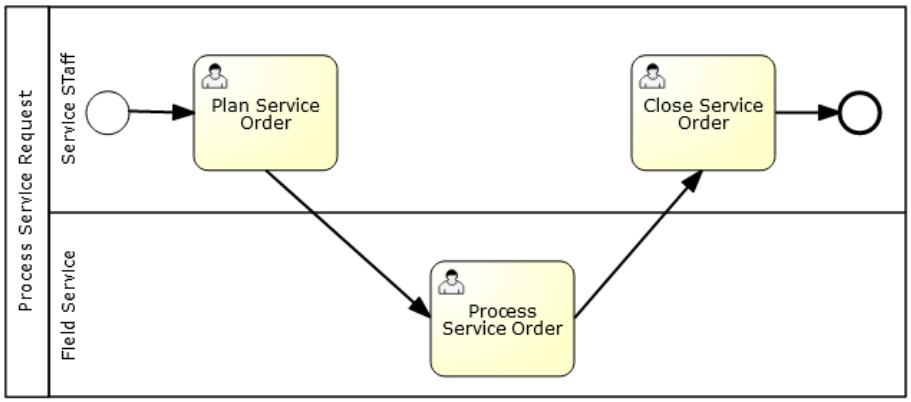
- Check Service Request
  - *Lane*: "Support Staff"
  - *Activity*: "Review"
  - *Expression When Completing the Activity*: see below
    Note: You must use the Fabasoft Cloud ID of your objects.

Check Service Request (*Expression When Completing the Activity*)

```
// Replace: reference and domain ID of your forms (the
// reference can be found in the properties of the form category)
import LASC_1_8075RELEASEFORMS@1.506;
if (object.COOSIGNATURE@1.1:lastsigned == COO.1.1001.1.198684) {
  // Replace: ID of your external organization "Customer"
  FSCFOLIOCLOUD@1.1001:ExtOrganization @extorgcustomer = COO.1.506.1.8143;
  STRING @sender = object.COOMAPI@1.1:mailsender;
  STRING @email = (coouser.FSCEXPEXT@1.1001:RegexCapture(@sender,
    "[a-zA-Z0-9+._-]+@[a-zA-Z0-9._-]+\\.[a-zA-Z0-9_-]+"))[0];
  // Creates a customer user object (if not existing)
  User @newcustomer = coouser.FSCORGMGMT@1.1001:CreateUser(@email,
    , , , , , , , @extorgcustomer, false);
  // Replace: ID of the form template "Service Order" (the template can be
  // found in the properties of the form)
  FSCUSERFORMS@1.1001:ContainerWithUserForm @template = COO.1.506.3.3527;
  // Creates a service order object
  FSCUSERFORMS@1.1001:ContainerWithUserForm @instance =
    @template.FSCTEMPLATEMGMT@1.1001:InstantiateTemplateObject();
  @instance.[#sodocuments] += object;
  @instance.[#socustomer] = @newcustomer;
  @instance.[#soactivity] = object.objname;
  // Replace: ID of the "Sevice Orders" instance
  FSCUSERFORMS@1.1001:RoomWithUserForm @room = COO.1.506.3.3539;
  @room.ObjectLock(true, true);
  // Adds the service order object to the "Service Orders"
  @room.[#soserviceorders] += @instance;
  // Removes the e-mail from the "Support Inbox"
  // Replace: ID of the "Support Inbox" instance
  FSCCLOUDMAIL@111.100:CloudMailInboxRoom @inbox = COO.1.506.3.3540;
  @inbox.ObjectLock(true, true);
  @inbox.FSCTEAMROOM@1.1001:ibrchildren -= object;
  // Starts the "Process Service Order" process
  // Replace: ID of the "Process Service Order" process (see next chapter)
  @instance.COOWF@1.1:StartProcessDiagram(COO.1.506.2.3543);
}
```

14.4.2 Process Service Order

Create the BPMN process "Process Service Order".



Pool

- *Applicable for*: "Service Order"

Lanes

- Service Staff
  - *Process Participant* > *Organizational Unit*: "Service Staff"
- Field Service
  - *Process Participant* > *Abstract Participant*: "Property of the Object" ("Service Technician")

Tasks

- Plan Service Order
  - *Lane*: "Service Staff"
  - *Activity*: "Edit"
  - *Expression When Completing the Activity*:
    ```
    object.ObjectLock(true, true);
    object.FSCFOLIO@1.1001:bostate = #FSCFOLIO@1.1001:StateInProgress;
    ```
- Process Service Order
  - *Lane*: "Field Service"
  - *Activity*: "Edit"
  - *Expression When Completing the Activity*: see below

Process Service Order (*Expression When Completing the Activity*)

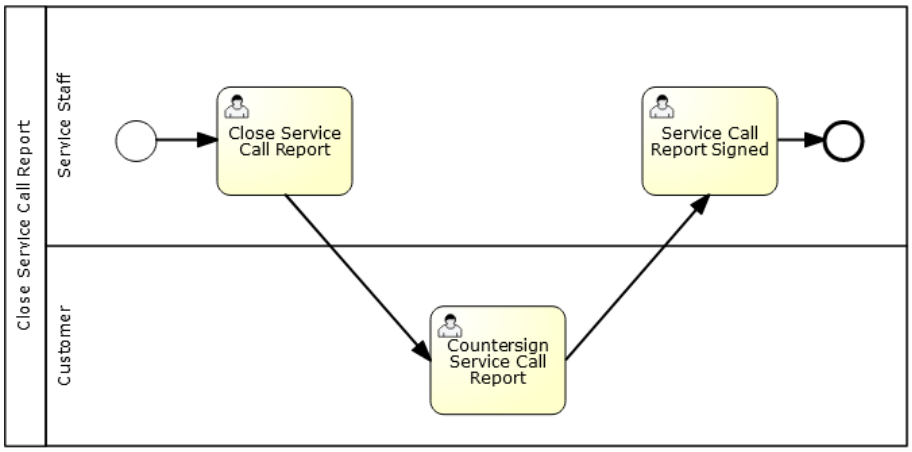
```
// Replace: reference and domain ID of your forms (the
// reference can be found in the properties of the form category)
import LASC_1_8075RELEASEFORMS@1.506;
object.ObjectLock(true, true);
object.FSCFOLIO@1.1001:bostate = #FSCFOLIO@1.1001:StateToVerify;
// Replace: ID of the "Service Call Report" Word template
COOMSOFFICE@1.1:WinWordObject @instance =
  COO.1.506.2.3535.FSCTEMPLATEMGMT@1.1001:InstantiateTemplateObject();
@instance.FSCFOLIO@1.1001:objfile = object;
// Adds the Word instance to the service order documents
object.[#sodocuments] += @instance;
// Resolves the text modules in the Word instance
@instance.FSCTEMPLATEMGMT@1.1001:ResolveTextModules();
// Updates the tables in the Word instance
@instance.FSCAPPTOOLS@1.1001:UpdateTables();
```

```
// Adds the first and the last image of the service order documents list to
// the Word instance
@instance.FSCAPPTOOLS@1.1001:UpdatePictures("Picture before service call",
object.[#sodocuments][IsUsable(#FSCWEBCONT@1.1001:ImageObject)][0]);
@instance.FSCAPPTOOLS@1.1001:UpdatePictures("Picture after service call",
object.[#sodocuments][IsUsable(#FSCWEBCONT@1.1001:ImageObject)][-1]);
// Starts the "Close Service Call Report" process
// Replace: ID of the "Close Service Call Report" process (see next chapter)
@instance.COOWF@1.1:StartProcessDiagram(COO.1.506.3.3544);
```

- Close Service Order
  - *Lane*: "Service Staff"
  - *Activity*: "Close File"
  - *Expression When Completing the Activity*:
    ```
    object.ObjectLock(true, true);
    object.FSCFOLIO@1.1001:bostate = #FSCFOLIO@1.1001:StateReleased;
    ```

14.4.3 Close Service Call Report

Create the BPMN process "Close Service Report". The sign activities are only available if there are digital signatures configured for your organization.



Pool

- *Applicable for*: "Service Call Report"
- *Object Class/Category of the File*: "Service Order"

Lanes

- Service Staff
  - *Process Participant* > *Organizational Unit*: "Service Staff"
- Customer
  - *Process Participant* > *Abstract Participant*: "Property of the File of the Object" ("Customer")

Tasks

- Close Service Call Report
  - *Lane*: "Service Staff"
  - *Activity*: "Sign Digitally"
- Countersign Service Call Report

    - *Lane*: “Customer”
    - *Activity*: “Sign Digitally”
- Service Call Report Signed
    - *Lane*: “Service Staff”
    - *Activity*: “Take Note”

14.5 Inbox

Configure the inbox “Support Inbox” (see chapter 14.4 “Processes”) with the following rule.

Inbox

- *Apply Rules Only for Top Level*: “Yes”

Rule

- *Name*: “Process Service Request”
- *Actions*: “Assign category ‘Service Request’”, “Start process ‘Check Service Request’”

14.6 Cloud Mail Import

In a Teamroom, create an a “Cloud Mail Import” named “Support Mail Inbox” which imports e-mails from your support mailbox.

“E-Mail Server” tab

- Enter mail server data.
- *Target Folder*: “Support Inbox”

“Automation” tab

- E-mails can be imported regularly and automatically.

14.7 Result

Having completed the model-based customizations, you can test the service process.

1. Make sure to have an e-mail in your mailbox.
2. Navigate to your “Support Mail Inbox” and execute the context menu command “Check Now”.
3. Navigate into your “Support Inbox”. Therein, the e-mails from your e-mail server must be available. The e-mails must be assigned the category “Service Request” (“General” tab) and the process “Check Service Request” (“Processes” tab) must be available.
4. Navigate into your worklist.
   Note: It is assumed that for the purpose of this test you made yourself the sole participant in all activities.
5. In the activity “Check Service Request”, do the task “Review”.
6. In the activity “Plan Service Order”, do the task “Open Properties”. In the field “Service Technician”, select yourself.
7. In the activity “Plan Service Order”, do the task “Finished”.

8. In the activity “Process Service Order”, do the task “Open Properties”. Check the option “Ball Bearing Lubricated”.
9. In the activity “Process Service Order”, do the task “Upload” two times to upload two images which serve as evidence.
10. In the activity “Process”, do the task “Finished”.
11. In the activity “Close Service Call Report”, do the task “Sign Digitally and Close”.
    Note: In this example, you must skip the activity “Close Service Order” for now since it must be done at the end.
12. In the activity “Countersign Service Call Report”, do the task “Sign Digitally”.
13. In the activity “Service Call Report Signed”, do the task “Take Note”.
14. In the activity “Close Service Order”, do the task “Close”. This will close the service order.

Now all processes are concluded. The finished service order can be found in “Service Orders”.