White Paper

Predefined Customization Points

2017 October Release

Fabasoft®

Contents

1 Introduction 5
2 Software Requirements 5
3 General 5
3.1 PreGUI 5
3.2 InitWithState 6
3.3 PostGUI 6
3.4 IncreaseOrdinal 7
3.5 ACLConfiguration 8
3.6 ImageTypeConfiguration 9
3.7 ContentConfiguration 9
3.8 CPSymbols 10
3.9 CPValidationExpression 11
3.10 CPContextExpressions 11
3.11 CPDocStateValidateConfig 12
3.12 CPRestrictClasses 13
3.13 FilterDispViewListAction 13
3.14 AggregationOverride 14
3.15 GetLogoContainer 15
3.16 CreatePortalConfiguration 15
3.17 CPAllowedAttrDef 16
3.18 ListOption 16
3.19 CPStateDisplay 18
3.20 CPGetArchivingConfig 18
4 Format 20
4.1 FormatValue 20
4.2 NameBuild 20
5 PDF Overview 21
5.1 TOCElementConfiguration 21
6 Room 22
6.1 CPGetRoomRoles 22
6.2 CPNotifiedRoomRoles 23
6.3 CPGetRoomPublishStates 24
6.4 CPSendRoomInvitation 24
6.5 CPGetRoomSecurity 25
6.6 CPChangeAllocationPermission 26
6.7 CPGetTargetRoomClasses 26

6.8 CPPublicConfigurationProperties 27
6.9 CPAutoCreateAppConfiguration 27
6.10 CPGetInitialConfigurationUsers 28
6.11 CPUseGlobalAppDashboard 29
6.12 CPTeamControl 29
6.13 CPExpandTeamControl 30
6.14 CPGETWBContainers 31
7 Transfer 32
7.1 AdditionalTransferObjects 32
7.2 AdditionalTransferParticipants 33
7.3 TransferContainer 33
7.4 TransferContainerLocation 34
7.5 FinishTransfer 34
7.6 ConcludeSecurity 35
8 Personnel File 36
8.1 CPInstanceACLs 36
8.2 InheritSecurityContext 36
9 Search 37
9.1 CPQuickSearchAction 37
9.2 CPQuickSearchAppearance 37
9.3 CPQuickSearchSuffix 38
9.4 CPMindbreezeQueryConstraints 39
10 Workflow 40
10.1 MetaParticipant 40
10.2 ProcessOwnership 41
10.3 InsertActivityDef 42

1 Introduction

The purpose of the app.ducx customization language is to define, customize and tailor your software component to project- or solution-specific requirements. Several customization points are defined by the base product itself. In this document these customization points are explained.

2 Software Requirements

System environment: All information contained in this document implicitly assumes a Microsoft Windows environment or a Linux environment.

Supported platforms: For detailed information on supported operating systems and software see the software product information on the Fabasoft distribution media.

3 General

3.1 PreGUI

The customization point `PreGUI` allows initializing values before the constructor form is displayed.

Syntax

```
customize PreGUI<objectclass> {
  steps = expression {...}
}
```

Description:

- *objectclass*
  The customization applies to objects of the defined object class.
- steps
  Defines a Fabasoft app.ducx Expression that is executed.

Example

app.ducx Object Model Language

```
instance ComponentState StateCaptured {
}
instance ComponentState StateApproved {
}
instance ComponentState StateRejected {
}
instance ComponentDocumentCategory PreGUICategory {
  dcshortform = "PreGui";
  dcstate = { StateCaptured,
              StateApproved,
              StateRejected }
}
```

app.ducx Customization Language

```
customize PreGUI<Incoming> {
  steps = expression {
    this.COOSYSTEM@1.1:ObjectLock(true, true);
    this.FSCFOLIO@1.1001:bostate = #StateCaptured;
    this.FSCFOLIO@1.1001:InitPreGUI();
```

```
    this.FSCFOLIO@1.1001:InitWithCategory(#PreGUICategory);
  }
}
```

Note: In the Fabasoft Folio Domain the customization gets defined in the template configuration.

3.2 InitWithState

The customization point `InitWithState` is a specialization of the customization point `PreGUI`.

Syntax

```
customize InitWithState<objectclass, state, category> {}
```

Description:

- *objectclass*
  The customization applies to objects of the defined object class.
- *state*
  Defines a state that is assigned to `FSCFOLIO@1.1001:bostate`.
- *category*
  Defines a category that is used for `FSCFOLIO@1.1001:InitWithCategory(#category)`.

Example

```
customize InitWithState<Incoming, StateCaptured, PreGUICategory> {}
```

Note: In the Fabasoft Folio Domain the customization gets defined in the template configuration.

3.3 PostGUI

The customization point `PostGUI` allows executing a Fabasoft app.ducx Expression after the user clicked “Next” or “Apply” of a constructor form.

Syntax

```
customize PostGUI<objectclass> {
  steps = expression {...}
}
```

Description:

- *objectclass*
  The customization applies to objects of the defined object class.
- steps
  Defines a Fabasoft app.ducx Expression that is executed.

Example

```
customize PostGUI<NoteObject> {
  steps = expression {
    this.COOSYSTEM@1.1:ObjectLock(true, true);
    this.COOSYSTEM@1.1:objsubject = "MySubject";
  }
```

```
}
```

Note: In the Fabasoft Folio Domain the customization gets defined in the template configuration.

3.4 IncreaseOrdinal

The customization point `IncreaseOrdinal` allows increasing a key numerator at `PostGUI`.

Syntax

```
customize IncreaseOrdinal<objectclass, property> {}
```

Description:

- *objectclass*
  The customization applies to objects of the defined object class.
- *property*
  Defines the numerator property that should be used.

Example

app.ducx Object Model Language

```
class ObjOrdinal : Document {
  KeyNumerator ordinal {
    /*
     * Specifies amount of preallocation
     */
    allocamount = 101;
    KeyEntryList<NUMERATOR@1.1001:objclass, keyattrlist> = {
      { ObjOrdinal, { documentyear } }
    }
  }
}
```

app.ducx Customization Language

```
customize IncreaseOrdinal<ObjOrdinal, ordinal> {}
```

Note: The parameter `alloc` has a default value of `true`. To create a numerator without preallocation you must explicitly specifiy `alloc = false`.

Note: In the Fabasoft Folio Domain the customization gets defined in the template configuration. This means that overrides objects from `FSCVTC@1.1001` for all applications responsible for creation of must be provided:

Example

```
// Application executed when an object is created via menu or
// button in an object list
override FSCVENV@1.1001:InitializeCreatedObject {
  variant objectclass {
    impl = FSCVTC@1.1001:InitializeCreatedObjectApp;
  }
}
// Application executed when an object is created inside of
// an object pointer property
override FSCVENV@1.1001:InitializeCreatedObjectDoDefault {
  variant objectclass {
    impl = FSCVTC@1.1001:InitializeCreatedObjectApp;
  }
```

```
}
// Application executed when an object is created from
// a template
override FSCVENV@1.1001:InitializeTemplateCreatedObject {
  variant objectclass {
    impl = FSCVTC@1.1001:InitializeCreatedObjectApp;
  }
}
```

3.5 ACLConfiguration

The customization point `ACLConfiguration` allows defining tenant specific ACLs that should be assigned to objects. The ACL is evaluated using following steps. The first step that returns an ACL defines the used ACL. Step 1 and 4 can be configured by the customization point.

1. Use a tenant specific ACL for an object, if a customization is defined for the corresponding object class (object class hierarchy gets considered).
   Trigger: `COOSYSTEM@1.1001:classdefaultacl`
2. Use an ACL that is defined for the object class (ACLs created in a corresponding tenant are preferred).
3. Use an ACL that is defined in the group of the user (*Default ACL for New Objects*, *ACL Objects*).
4. Use a tenant specific default ACL for an object, if a customization is defined for the corresponding object class (object class hierarchy gets considered).
   Trigger: `COOSYSTEM@1.1001:objaclobj`
5. Hard-coded: If the steps before do not return an ACL, finally one of these ACLs is used based on the object class: “Default ACL”, “ACL for Administration Objects” or “ACL for Developer Objects”.

Syntax

```
customize ACLConfiguration<cfgobjclass, trigger> {
  acl = ...
}
```

Description:

- `cfgobjclass`
  The customization applies to objects of the defined object class.
- *`trigger`*
  If step 1 should be customized, `COOSYSTEM@1.1001:classdefaultacl` has to be used. If step 4 should be customized, `COOSYSTEM@1.1001:objaclobj` has to be used.
- `acl`
  Defines the ACL that should be assigned.

Example

```
customize ACLConfiguration<Folder, classdefaultacl> {
  acl = COOSYSTEM@1.1:DefaultGlobalACL;
}
```

Note: In the Fabasoft Folio Domain the customization gets defined in the administration configuration.

3.6 ImageTypeConfiguration

The customization point `ImageTypeConfiguration` allows defining parameters for thumbnails and preview images.

Syntax

```
customize ImageTypeConfiguration<imgclass, imgtype> {
  imgwidth = ...
  imgheight = ...
  imgformat = ...
  imgpages = ...
  imgdstattrdef = ...
  imgdefaultthumbexpr = expression {...}
}
```

Description:

- *imgclass*
  The customization applies to objects of the defined object class.
- *imgtype*
  Defines whether the customization applies to thumbnails (“tn”) or preview images (“pv”).
- `imgwidth`
  Defines the maximum width in pixel of the displayed image (e.g. 512).
- `imgheight`
  Defines the maximum height in pixel of the displayed image (e.g. 256).
- `imgformat`
  Defines the format of the displayed image (default: “jpg”; possible values: “gif”, “png”).
- `imgpages`
  Defines how many pages of a document should be available as image.
- `imgdstattrdef`
  Defines the property where the image should be cached (e.g. `COODESK@1.1:objthumbnailimage` or `COODESK@1.1:objpreviews`).
- `imgdefaultthumbexpr`
  Defines an expression that is used to evaluate a fallback image, if the actual image is not available (e.g. `cooobj.FSCVENV@1.1001:GetObjectImageFallback(imgtype)`).

Example

```
customize ImageTypeConfiguration<Object, "tn"> {
  imgwidth = 512;
  imgheight = 256;
  imgdstattrdef = objthumbnailimage;
  imgdefaultthumbexpr = expression {
    cooobj.GetObjectImageFallback(imgtype)
  }
}
```

Note: In the Fabasoft Folio Domain the customization gets defined in the conversion configuration.

3.7 ContentConfiguration

The customization point `ContentConfiguration` defines in which property the desired content can be found (e.g. content for the thumbnail or preview image generation).

Syntax

```
customize ContentConfiguration<contclass> {
  contsrcexpr = expression {...}
}
```

Description:

- *contclass*
  The customization applies to objects of the defined object class.
- contsrcexpr
  Defines the expression that is used to evaluate the desired content (e.g. for the thumbnail and preview image generation or for conversion tasks).

Example

```
customize ContentConfiguration<ContentObject> {
  contsrcexpr = expression { cooobj.classinitcont }
}
```

Note: In the Fabasoft Folio Domain the customization gets defined in the conversion configuration.

3.8 CPSymbols

The customization point `CPSymbols` is used to define the symbol of objects from a certain object class.

Syntax

```
customize CPSymbols<cfgobjclass> {
  cfgicexpression = expression {...}
  objmicon = ...;
}
```

Description:

- *cfgobjclass*
  The customization applies to objects of the defined object class.
- cfgicexpression
  If this expression returns “true”, the symbol `objmicon` is used. Alternatively the expression itself can return a symbol to be used.
- objmicon
  Defines the symbol that is assigned to `COODESK@1.1:Symbol`.

Example

```
customize CPSymbols<User> {
  cfgicexpression = expression {
    return cooobj.useractive;
  }
  objmicon = SymbolUserAccepted;
}
```

Note: In the Fabasoft Folio Domain the customization gets defined in the administration configuration.

Note: This customization point gets evaluated in the `FSCCONFIG@1.1001:MethodGenericIconGet` method, which is used to implement the `COODESK@1.1001:AttrObjMiniIconGet` action for the appropriate object class *`cfgobjclass`*.

3.9 CPValidationExpression

The customization point `CPValidationExpression` is used for the validation of an entered value. The customization point gets evaluated in the `FSCCONFIG@1.1001:MethodValidateSet` method, which can be called in the `FSCCONFIG@1.1001:AttrValidateSet` action.

The local scope contains the current object and the global scope contains a dictionary with the old value and the new value.

Syntax

```
customize CPValidationExpression<cfgobjclass, trigger, context> {
  cfgexpression = expression {...}
  errorobject = ...;
}
```

Description:

- *`cfgobjclass`*
  The customization applies to objects of the defined object class.
- *`trigger`*
  The key for the evaluation of the output parameter.
- *`context`*
  The optional key for the evaluation of the output parameter.
- `cfgexpression`
  If this expression returns “true”, the error (`errorobject`) is displayed. Alternatively the expression itself can display an error.
- `errorobject`
  The error message that is displayed in case of an error.

Example

```
customize CPValidationExpression<User, DialogUserPasswordForgot, objname> {
  cfgexpression = expression {
    return true;
  }
  errorobject = ErrActivateUserFound;
}
```

Note: In the Fabasoft Folio Domain the customization gets defined in the administration configuration.

3.10 CPContextExpressions

The customization point `CPContextExpressions` returns an expression, depending on the input parameters `trigger` and `context`.

The customization point gets evaluated in the `FSCCONFIG@1.1001:EvaluateExpression` action. This action has to be called for the appropriate object class (`cfgobjclass`) with the input parameters `trigger` and optionally `context`.

Syntax

```
customize CPContextExpressions<cfgobjclass, trigger, context> {
  cfgexpression = expression {...}
}
```

Description:

- *cfgobjclass*
  The customization applies to objects of the defined object class.
- *trigger*
  The key for the evaluation of the output parameter.
- *context*
  The optional key for the evaluation of the output parameter.
- cfgexpression
  The expression to be returned.

Example

```
customize CPContextExpressions<User, DialogUserPasswordForgot, objname> {
  cfgexpression = expression {
    return true;
  }
}
```

Note: In the Fabasoft Folio Domain the customization gets defined in the administration configuration.

3.11 CPDocStateValidateConfig

The customization point `CPDocStateValidateConfig` is used for validating state changes of documents.

To evaluate the customization point, the `FSCFOLIO@1.1001:ValidateDocStateChange` action has to be called for an object of the appropriate object class (`dsvcobjclass`). By means of the state change (`dsvcoldstate`, `dsvcnewstate`) this action determines the respective expression (`dsvcvalidateexpr`) and evaluates it. Optionally the defined signature type (`dsvcsigntype`) can be executed.

Syntax

```
customize CPDocStateValidateConfig<dsvobjclass, dsvcoldstate, dsvcnewstate> {
  dsvcvalidateexpr = expression {...}
  dsvcsigntype = ...;
}
```

Description:

- *dsvcobjclass*
  The customization applies to objects of the defined object class.
- *dsvcoldstate*
  The state of the document, before the state change.
- *dsvcnewstate*
  The state of the document, after the state change.

- `dsvcvalidateexpr`
  An optional Fabasoft app.ducx Expression that has to be apply.
- `dsvcsigntype`
  Defines the *Signature Type* (`FSCFOLIO@1.1001:dsvcsigntype`) that has to be used after the state changed.

Example

```
customize CPDocStateValidateConfig<Case, DS_EDIT, DS_CLOSED> {
  dsvcvalidateexpr = expression {
    return true;
  }
  dsvcsigntype = SIGN_CLOSE;
}
```

Note: In the Fabasoft Folio Domain this customization gets defined in the Folio configuration.

3.12 CPRestrictClasses

The customization point `CPRestrictClasses` is used to restrict createable and searchable classes. In an expression all restricted classes are returned.

Syntax

```
customize CPRestrictClasses<cfgobjclass, cfgattrdef, cfgcamode> {
  cfgcpexpression = expression {...}
}
```

Description:

- *`cfgobjclass`*
  The customization applies to objects of the defined object class.
- *`cfgattrdef`*
  The property, to which the customization applies.
- *`cfgcamode`*
  The validation mode for allowed classes.
- *`cfgcpexpression`*
  The expression to be returned. This expression contains restricted classes.

Example

```
customize CPRestrictClasses<Object, objchildren, null> {
  cfgcpexpression = expression {
    return [#AppProject];
  }
}
```

Note: For the expression, the values `attrdef`, `mode`, `restriction`, `allowedclasses`, `notallowedclasses` are available in the global scope.

3.13 FilterDispViewListAction

The customization point `FilterDispViewListAction` determines the filter action for the display view settings for the given object class.

Syntax

```
customize FilterDispViewListAction<objclass> {
  filteraction = expression {...}
}
```

Description:

- *objclass*
  The customization applies to objects of the defined object class.
- *filteraction*
  This action is used to verify or override the resulting display view list. The action uses the prototype `COODESK@1.1:FilterDispViewListPrototype`.

Example

app.ducx Customization Language

```
customize FilterDispViewListAction<Folder> {
  filteraction = expression {
    return #MYCUSTOMIZE@1.1065:FolderDispViewListFolder
  }
}
```

app.ducx Use Case Language

```
usecase FolderDispViewListFolder(parameters as FilterDispViewListPrototype) {
  variant Folder {
    impl = expression {
      if (writelocation >= 0) {
        // write columns: prevent saving settings
        throw #COOSTERR_CANCEL;
      }
      else {
        // read columns: fixed set of display columns
        DisplayColumnList[] mycolumns;
        mycolumns += coort.CreateAggregate(#DisplayColumnList);
        mycolumns += coort.CreateAggregate(#DisplayColumnList);
        mycolumns += coort.CreateAggregate(#DisplayColumnList);
        mycolumns[0].dispattribute = #objname;
        mycolumns[1].dispattribute = #objcreatedby;
        mycolumns[2].dispattribute = #objaclobj;
        displaylist.dispcolumns = mycolumns;
      }
    }
  }
}
```

3.14 AggregationOverride

The customization point `AggregationOverride` replaces an aggregation action for the given object class and is available in the software component `FSCVENVUI@1.1001`.

Syntax

```
customize AggregationOverride<objclass, attrdef, aggract> {
  aggroverride = expression {...}
}
```

Description:

- *objclass*
  The customization applies to objects of the defined object class.

- *attrdef*
  The property, to which the customization applies.
- *aggract*
  The aggregation action, which is to be overridden.
- *aggroverride*
  This action is used to override the aggregation action. The action uses the prototype `COODESK@1.1:AggregationPrototype`.

Example

app.ducx Customization Language

```
customize AggregationOverride<Object, objcontsize, GetSum> {
  aggroverride = expression {
    //calculates the sum of the values and adds a "KB" suffix
    return #FSCVENVUI@1.1001:GetSumContSize
  }
}
```

3.15 GetLogoContainer

The customization point `GetLogoContainer` defines which logo is displayed based on the object class of the selected object.

Syntax

```
customize GetLogoContainer<objclass> {
  cont = expression {...}
}
```

Description:

- *objclass*
  The customization applies to objects of the defined object class.
- *cont*
  Defines a Fabasoft app.ducx Expression to calculate the object that contains the logo (`COODESK@1.1:objlogoimage`) that should be displayed.

Example

app.ducx Customization Language

```
customize GetLogoContainer<Object> {
  cont = expression {
    if (cooobj.HasClass(#TeamRoom)) {
      cooobj;
    }
    else {
      cooobj.objteamroom != null ? cooobj.objteamroom : null;
    }
  }
}
```

3.16 CreatePortalConfiguration

The customization point `CreatePortalConfiguration` defines the template for creating a portal entry if an object is dragged to the portal list.

Syntax

```
customize CreatePortalConfiguration<cfgobjclass> {
  portalentry = ….;
}
```

Description:

- *cfgobjclass*
  The customization applies to objects of the defined object class.
- *portalentry*
  Defines the template to be used to create the portal entry.

Example

app.ducx Customization Language

```
customize CreatePortalConfiguration<Object> {
  portalentry = PaneDesk;
}
```

3.17 CPAllowedAttrDef

The customization point `CPAllowedAttrDef` allows redefining object pointer properties in the current domain for checking and filtering allowed object classes.

Syntax

```
customize CPAllowedAttrDef<cfgobjclass,cfgattrdef,cfgcamode> {
  cfgrefattrdef = ….;
  cfgexpression = ….;
}
```

Description:

- *cfgobjclass*
  The customization applies to objects of the defined object class.
- *cfgattrdef*
  References the object pointer property definition to be redefined.
- *cfgcamode*
  Defines the mode for checking or filtering allowed objectclasses.
- *cfgrefattrdef*
  Defines the new object pointer property for checking or filtering allowed object classes.
- *cfgexpression*
  Conditional expression to be executed to check if the replace has to be done.

Example

app.ducx Customization Language

```
customize CPAllowedAttrDef<Story,objchildren,null> {
  cfgrefattrdef = scrumdocuments;
}
```

3.18 ListOption

The customization point `ListOption` allows overriding the SimList control parameters.

Syntax

```
customize ListOption<objclass,attrdef> {
  createapplication = expression {...}
  createsymbol = expression {...}
  createtext = expression {...}
  searchtext = expression {...}
  typetext = expression {...}
  showicon = expression {...}
  showunderline = expression {...}
  wildcardsearch = expression {...}
  separator = expression {...}
  listseparator = expression {...}
}
```

Description:

- *objclass*
  The customization applies to objects of the defined object class.
- *attrdef*
  References the object pointer property definition to be redefined.
- *createapplication*
  Defines an expression to get the application for creating objects in the control.
- *createsymbol*
  Defines an expression to get the symbol for creating objects in the control.
- *createtext*
  Defines an expression to get the description for creating objects in the control.
- *searchtext*
  Defines an expression to get the search text for creating objects in the control.
- *typetext*
  Defines an expression to get the data entry instruction of the control.
- *showicon*
  Defines an expression to set a symbol for an entry.
- *showunderline*
  Defines an expression to set an unterline representation for an entry.
- *wildcardsearch*
  Defines an expression to allow wild card search in the control.
- *separator*
  Defines an expression to get the string that should be used between the objects.
- *listseparator*
  Defines an expression to get the string that should be used between the objects in cells.

Example

app.ducx Customization Language

```
customize ListOption<TransmissionLog,tlinvitedusers> {
  createapplication = expression {
    return #CreateObjectApp;
  }
  createsymbol = expression {
    return (cootx.isCreated(cooobj) ? #SymbolAdd : #MiniIconObjectCreate );
  }
  createtext = expression {
    return (cootx.isCreated(cooobj) ? #StrAddFavorite : #StrCreate);
  }
  typetext = expression {
```

```
    return #StrTypeText;
  }
  showicon = expression {
    return cootx.isCreated(cooobj);
  }
  showunderline = expression {
    return cooobj.tlinvitationcategory == #MC_TeamRoom;
  }
}
```

For detailed information on the control parameters of the `FSCSIMLIST@1.1001:ListOption` control refer to [Faba10c]

3.19 CPStateDisplay

The customization point `CPStateDisplay` allows defining visualizations for objects depending on the object state.

Syntax

```
customize CPStateDisplay<cfgobjclass> {
  cfgstatedisplayexpression = expression {...}
}
```

Description:

- *cfgobjclass*
  The customization applies to objects of the defined object class.
- cfgstatedisplayexpression
  Defines an expression to get the `FSCVAPP@1.1001:ObjectStateDisplay`.

Example

app.ducx Customization Language

```
customize CPStateDisplay<ContentObject> {
  cfgstatedisplayexpression = expression {
    Content $cont;
    object $obj = this.object;
    $obj.DirectAttributeGet(#content, &$cont);
    if (null != $cont) {
      EncryptionKind $contencryption = $cont.contencryption;
      if (null != $contencryption) {
        string $encryptstr=#EncryptionKind.typeenumvalues[typeenumval ==
          :>$contencryption].GetAttributeString(cootx, #typeenumtext);
        ObjectStateDisplay $encdisplaystate = coort.CreateAggregate(#ObjectStateDisplay);
        if ($encdisplaystate) {
          $encdisplaystate.objectstatedisplaysymbol = #UISymbolEncrypt;
          $encdisplaystate.objectstatedisplaydescription=encryptstr;
        }
        return $encdisplaystate;
      }
    }
  }
}
```

3.20 CPGetArchivingConfig

The customization point `CPGetArchivingConfig` allows simple archiving of a property. The archives will be added to a property and the objects will be moved into the archives. To start the archiving `ObjectListSetArchiveWrapper` has to be called. It is usually called by the set action of the property that should be archived (`attrdef`).

Syntax

```
customize CPGetArchivingConfig <cfgobjclass, attrdef > {
  cfgobjstomove = expression {...}
  cfgtargetobj = expression {...}
  cfgtargetobjattrdef = expression {...}
  cfgarchiveclass = expression {...}
  cgfarchivespan = expression {...}
  cfgarchiveattrdef = expression {...}
}
```

Description:

- *cfgobjclass*
  The customization applies to objects of the defined object class.
- *attrdef*
  Defines the property that should be archived.
- *cfgobjstomove*
  Defines the objects that should be moved into the archives.
- *cfgtargetobj*
  Defines the object where the *cfgobjstomove* should be archived
- *cfgtargetobjattrdef*
  Defines the property of the *cfgtargetobj* that should store the archives
- *cfgarchiveclass*
  Defines the object class of the archive. Has to be a derived form DefaultArchive
- *cgfarchivespan*
  Defines if new archives should be created yearly, quarterly or monthly
- cfgarchiveattrdef
  Defines the date property for evaluating the archive for an object (e. g. objcreated, objmodifiedat). Has to be a datetime

Example

app.ducx Customization Language

```
customize CPGetArchivingConfig<ProjectReport, prreports> {
      cfgobjstomove = expression {
        datetime limit = coonow;
        limit.month -= 1;
        return cfgobjstoarchive[objcreated <= limit];
      }

      cfgtargetobj = expression { cooobj };

      cfgtargetobjattrdef = expression { #prreports };

      cfgarchiveclass = expression { #ScrumReportArchive };

      cgfarchivespan = expression { ArchiveSpan(AS_QUARTER) };

      cfgarchiveattrdef = expression { # objcreated };
    }
```

4 Format

4.1 FormatValue

The customization point `FormatValue` allows formatting values. For the defined object class, the action `COOSYSTEM@1.1:ObjectPrepareCommit` will be overwritten implicitly.

Syntax

```
customize FormatValue<objectclass, trigger> {
  build = expression {...}
}
```

Description:

- *objectclass*
  Defines the object class of the object that contains the value that should be formatted.
- *trigger*
  Defines the property that should be formatted.
- build
  Defines a Fabasoft app.ducx Expression that formats the value.

Example

```
customize FormatValue<Project, objname> {
  build = expression {
    this.boshortform + " " + Format(this.recnumber, "0000")[4] + " - "
  }
}
```

Note: In the Fabasoft Folio Domain the customization gets defined in the administration configuration.

4.2 NameBuild

The customization point `NameBuild` is used to format values. This customization point cannot be used when the customization point `FormatValue` is used in a base class.

Syntax

```
customize NameBuild<cfgobjclass> {
  properties = {...}
  build = expression {...}
  format = stringvalue
  namefixed = booleanvalue
}
```

Description:

- *cfgobjclass*
  The customization applies to objects of the defined object class.
- *properties*
  The properties, to which the customization applies and used in the `build` expression.

- *build*
  Defines a Fabasoft app.ducx Expression that formats the value.
- *format*
  Defines a string object with a multilingual name.
- *namefixed*
  Defines if the value is editable.

Example

```
customize NameBuild <Account> {
  properties = {
    accountnumber,
    usersurname
  }
  build = expression {
    return cooobj.accountnumber + " / " + cooobj.usersurname;
  }
  namefixed = true;
}
```

Note: To use this customization point with Fabasoft domains until 2012 Summer Release, it is necessary to implement `FSCCONFIG@1.1001:EvaluateGenericNameBuild` for the class defined in `cfgobjclass` with *FSCCONFIG@1.1001:MethodGenericNameBuild*.

5 PDF Overview

5.1 TOCElementConfiguration

The customization point `FSCPDFGEN@1.1001:TOCElementConfiguration` defines the compound elements and their depending objects that should be displayed in the table of contents of the PDF overview.

The customization point `FSCPDFGEN@1.1001:TOCElementConfiguration` is evaluated in the use case “View as PDF” when the document definition `FSCPDFGEN@1.1001:TOCDocumentDefinition` is configured for the target object of the use case.

Syntax

```
customize TOCElementConfiguration<tocecclass> {
  tocecincludeexpr = expression {...}
  tocecchildattrdefs = objectlist
}
```

Description:

- *tocecclass*
  The customization applies to objects of the defined object class.
- `tocecincludeexpr`
  Defines an expression to check if the current object should be included in the PDF overview. Note: This parameter is only evaluated for the root entry when it is found for the first time in the table of contents. To include the root element self and not only it’s children the expression must return true.
- `tocecchildattrdefs`
  Defines the properties where the depending objects are located.

In the expression following additional variables are available:

- *toceroot*
  Container object for which the table of contents should is generated. This parameter is normally used to display the root element in separate ways when it is contained multiple times in the table of contents.

Example

```
customize TOCElementConfiguration<Folder> {
  tocecincludeexpr = expression {
    /*
     * Use only objects that are in the same room as the current root
     */
    return tocecroot.GetObjectTeamRoom() == cooobj.GetObjectTeamRoom();
  }
  tocecchildattrdefs = {
    objchildren
  }
}
customize PDFOverviewConfiguration<Folder> {
  pdfovconcernedexpr = expression {
    return [this, GetObjectList(, GetChildrenAttrDef(), , true)];
  }
  pdfovdocdef = TOCDocumentDefinition;
}
```

Note: Reference the software component `FSCCONV@1.1001` in your Fabasoft app.ducx project for this customization point together with `FSCPDFGEN@1.1001`.

6 Room

6.1 CPGetRoomRoles

The customization point `FSCTEAMROOM@1.1001:CPGetRoomRoles` defines the list of available room roles and the default role for new `FSCTEAMROOM@1.1001:Room` members.

The customization point `FSCTEAMROOM@1.1001:CPGetRoomRoles` is evaluated on several points of the `FSCTEAMROOM@1.1001:Room` business logic, e.g. on calculating the room roles to be shown in the team control.

The `roles` list is ordered by room role priority. The first list entry is the role with highest priority, while the last list entry is the role with the lowest priority.

Syntax

```
customize CPGetRoomRoles<objclass> {
  roles = expression {…}
  default = expression {…}
  multipleroles = boolean
}
```

Description:

- *objclass*
  The customization applies to objects of the defined object class.
- `roles`
  Defines an expression to get the list of available room roles. The return value is an object list of type `FSCTEAMROOM@1.1001:RoomRole[]`.

- `default`
  Defines an expression to get the default role for adding new team members. The return value is an object pointer to a `FSCTEAMROOM@1.1001:RoomRole` object.
- `multipleroles`
  Defines if a team member can have multiple room roles at the same time. Default value is false.

Example

```
customize CPGetRoomRoles<TeamRoom> {
  roles = expression {
    return [#RoleTeamRoomOwner, #RoleTeamRoomFullControl,
            #RoleTeamRoomCanChange, #RoleTeamRoomReadOnly];
  }
  default = expression {
    return #RoleTeamRoomCanChange;
  }
  multipleroles = false;
}
```

6.2 CPNotifiedRoomRoles

The customization point `FSCTEAMROOM@1.1001:CPNotifiedRoomRoles` defines the list of room roles which should be notified by email in some use cases.

The customization point `FSCTEAMROOM@1.1001:CPNotifiedRoomRoles` for example is evaluated when an organization member is excluded from an cloud organization and the member currently participates in the current room.

The `roles` can also include room roles from the app configuration.

Syntax

```
customize CPNotifiedRoomRoles<objclass, usecase> {
  roles = expression {…}
}
```

Description:

- *`objclass`*
  The customization applies to objects of the defined object class.
- *`usecase`*
  The usecase for which the notified room roles should be evaluated.
- `roles`
  Defines an expression to get the list of room roles which should be notified by email..

Example

```
customize CPNotifiedRoomRoles<Room> {
  roles = expression {
    return cooobj.GetRoomRoles(true);
  }
}

customize CPNotifiedRoomRoles<ScrumCenter> {
  roles = expression {
    return [#RoleScrumAdministrator];
```

```
  }
}

customize CPNotifiedRoomRoles<ScrumProject> {
  roles = expression {
    return [#RoleScrumAdministrator];
  }
}
```

6.3 CPGetRoomPublishStates

The customization point `FSCTEAMROOM@1.1001:CPGetRoomPublishStates` defines the list of available publish states for a `FSCTEAMROOM@1.1001:Room` class.

Syntax

```
customize CPGetRoomPublishStates<objclass> {
  publishstates = expression {…}
}
```

Description:

- *objclass*
  The customization applies to objects of the defined object class.
- publishstates
  Defines an expression to get the list of available publish states. The return value is an object list of type `FSCTEAMROOM@1.1001:RoomPublishState[]`.
- default
  Defines an expression to get the default publish state when a new room is created.

Example

```
customize CPGetRoomPublishStates<TeamRoom> {
  publishstates = expression {
    return [#TeamRoomTeamOnly, #TeamRoomPublicLookup, #TeamRoomPublicClearance];
  }
  default = expression {
    return #TeamRoomTeamOnly;
  }
}
```

6.4 CPSendRoomInvitation

The customization point `FSCTEAMROOM@1.1001:CPSendRoomInvitation` defines if invitation e-mails for an `FSCTEAMROOM@1.1001:Room` should be sent.

Syntax

```
customize CPSendRoomInvitation<objclass> {
  expr = expression {...}
}
```

Description:

- *objclass*
  The customization applies to objects of the defined object class.

- `expr`
  Defines an expression to get a Boolean if the invitation e-mail should be sent.

Example

```
customize CPSendRoomInvitation<TeamRoom> {
  expr = expression {
    return true;
  }
}
```

6.5 CPGetRoomSecurity

The customization point `FSCTEAMROOM@1.1001:CPGetRoomSecurity` is used to evaluate the security context for `FSCTEAMROOM@1.1001:Room` child objects depending on a particular `FSCTEAMROOM@1.1001:Room` use case.

The use cases are described by instances of the object class `FSCCONFIG@1.1001:Context`.

Currently these context objects are used: `FSCTEAMROOM@1.1001:CtxApplyRoom`, `FSCTEAMROOM@1.1001:CtxChangePublishState`, `FSCTEAMROOM@1.1001:CtxMoveToWastebasket` as well as `FSCTEAMROOM@1.1001:CtxRestoreFromWastebasket`.

The customization point `FSCTEAMROOM@1.1001:CPGetRoomSecurity` is evaluated if a use case according to one of these contexts is executed.

Syntax

```
customize CPGetRoomSecurity<objclass, roomobjclass, usecase, room> {
  seccontext = expression {...}
}
```

Description:

- *objclass*
  The customization applies to objects of the defined object class.
- *roomobjclass*
  The object class of the room to which the current object is assigned to.
- *usecase*
  Defines the use case for security context evaluation
- *room*
  The room of the current object.
- `seccontext`
  Defines an expression to get the security context. A security context is an aggregate of type `FSCFOLIO@1.1001:SecurityContext`.

Example

```
customize CPGetRoomSecurity <Object, TeamRoom, CtxApplyRoom> {
  roles = expression {
    SecurityContext ctx = null;
    if (room && room.HasClass(#Room)) {
      Object aclref = room.objaclref;
      ctx = {
        objaccdef : room.objaccdef,
        objaclobj : room.objaclobj),
```

```
        objaclref = :>aclref ? :>room : null
      };
    }
    return ctx;
  }
}
```

6.6 CPChangeAllocationPermission

The customization point `FSCTEAMROOM@1.1001:CPChangeAllocationPermission` is used to define the access types which are needed on the source room to change the allocation of an object another room. The customization point is evaluated by the action `FSCTEAMROOM@1.1001:HasRoomChangeAllocationAccess`.

The use cases are described by instances of the object class `FSCCONFIG@1.1001:Context`.

Syntax

```
customize CPChangeAllocationPermission<objclass> {
  requiredpermission = expression {...}
}
```

Description:

- *objclass*
  The object class of the room to which the current object is assigned to.
- *requiredpermission*
  Access type required to change the room.

Example

```
customize CPChangeAllocationPermission <TeamRoom> {
  requiredpermission = expression {
    return #AccTypeChangeSec;
  }
}
```

6.7 CPGetTargetRoomClasses

The customization point `FSCTEAMROOM@1.1001:CPGetTargetRoomClasses` defines if the room assignment of an object, which is referenced by more than one room, can be changed.

Syntax

```
customize CPGetTargetRoomClasses <Sourceclass, Objclass, SOFTWARECOMPONENT> {
  targetclasses = Targetroom
}
```

Description:

- *sourceroomclass*
  The object class of the object's current room
- *objclass*
  The object's class

- *component*
  Reference of the customizing software component
- *targetclasses*
  Possible object classes of target rooms.

Example

```
customize CPGetTargetRoomClasses<TeamRoom, CompoundObject , FSCTEAMROOM@1.1001> {
  targetclasses = TeamRoom;
}

customize CPGetTargetRoomClasses<TeamRoom, ContentObject, FSCSCRUM@1.1001> {
  targetclasses = {RootObject, Folder}
}
```

6.8 CPPublicConfigurationProperties

The customization point `FSCTEAMROOM@1.1001:CPPublicConfigurationProperties` defines a list of properties which can be read from the app room or app configuration room without access permissions to the app room or app configuration room.

Syntax

```
customize CPPublicConfigurationProperties <sourcroomeclass> {
  attributes = attributes
}
```

Description:

- *sourceroomclass*
  The object class of the object's current room
- *attributes*
  The configuration properties of the app configuration or app room which are allowed to read without access permissions

Example

```
customize CPPublicConfigurationProperties<PersonnelFileShelf> {
  attributes = {
    pfsimportmasterfiledata,
    pfswatermarktext,
    pfswatermarkimage
  }
}

customize CPGetTargetRoomClasses<TeamRoom, ContentObject, FSCSCRUM@1.1001> {
  targetclasses = {RootObject, Folder}
}
```

6.9 CPAutoCreateAppConfiguration

The customization point `FSCTEAMROOM@1.1001:CPAutoCreateAppConfiguration` defines if an app configuration room should be created automatically for an organization when the app which supplies the app configuration is licensed.

When the app configuration room is created automatically the users returned by the customization point `FSCTEAMROOM@1.1001:CPInitalConfigurationUsers` will be the initial administrators of the app configuration room. Otherwise these users will get an welcome screen entry to create an app configuration room.

Syntax

```
customize CPAutoCreateAppConfiguration <objclass> {
  autocreate = boolean
}
```

Description:

- *objclass*
  The object's class
- *autocreate*
  Defines if the room should be created automatically or if an welcome screen event should be displayed

Example

```
customize CPAutoCreateAppConfiguration<ScrumCenter> {
  autocreate = false;
}
```

6.10 CPGetInitialConfigurationUsers

The customization point `FSCTEAMROOM@1.1001:CPGetInitalConfigurationUsers` defines the users which are used as initial administrators for automatically created app configuration rooms or for the welcome screen event which requests to create an app configuration room.

Syntax

```
customize CPGetInitialConfigurationUsers <objclass > {
  configusers  = expression {...}
}
```

Description:

- *objclass*
  The object class of the object's current room
- *group*
  The organization for which the app configuration room should be created is available in the local scope of the customization point
- *configusers*
  The users which should get request.

Example

```
customize CPGetInitialConfigurationUsers<AppConfigurationRoom> {
  configusers  = expression {
    return coouser.ExpandGroupMembers(group.objowner +
                                      group.objsecsecurity +
                                      group.objsecchange);
  }
}
```

6.11 CPUseGlobalAppDashboard

The customization point `FSCTEAMROOM@1.1001:CPUseGlobalAppDashboard` defines if one global app dashboard should be created for each user. Otherwise a separate app dashboard is created for each app configuration room the user was added to. By default an app dashboard is created for each app configuration room instance.

Syntax

```
customize CPUseGlobalAppDashboard <objclass > {
  useglobal = boolean
  hideappconfiguration = boolean
}
```

Description:

- *objclass*
  The object class of the app configuration room for which an app dashboard should be created
- *useglobal*
  The return value defines if a global or separate app dashboards should be created
- *hideappconfiguration*
  The app configuration room is not shared to the desk of the app administrator when he receives the license. By default the app configuration room is shared to the desk.

Example

```
customize CPUseGlobalAppDashboard<AppConfigurationRoom> {
  useglobal = false;
  hideappconfiguration = false;
}
customize CPUseGlobalAppDashboard<ScrumCenter> {
  useglobal = true;
  hideappconfiguration = true;
}
```

6.12 CPTeamControl

The customization point `FSCTEAMROOM@1.1001:CPTeamControl` is used to evaluate texts and symbols displayed in the team control. The team control is displayed for all room classes, so the configuration can be changed for each class derived from `FSCTEAMROOM@1.1001:Room`. A default configuration for the class `FSCTEAMROOM@1.1001:Room` is already supplied.

Syntax

```
customize CPTeamControl<objclass> {
  placeholdertext = expression {...}
  noresulttext = expression {...}
  createtext = expression {...}
  createhovertext = expression {...}
  createsymbol = expression {...}
  createapplication = expression {...}
  excludedattrdefs = expression {...}
  disablecreate = expression {...}
}
```

Description:

- *objclass*
  The customization applies to objects of the defined object class.

- `placeholdertext`
  Defines an expression to get the string that will be displayed as placeholder in the select box to find an add new team members.
- `noresulttext`
  Defines an expression to get the string that will be displayed in the select box when no contacts matching the search criteria were found..
- `createtext`
  Defines an expression to get the string that will be displayed in the select box to create a new contact.
- `createhovertext`
  Defines an expression to get the string that will be show as hover text of the create button in the xml dispatcher.
- `createsymbol`
  Defines an expression to get the symbol that will be displayed beside the create text. In the xml dispatcher it is used as symbol of the create button too.
- `createapplication`
  Defines an expression to get the application that will executed when a new contact should be created..
- `excludedattrdefs`
  Defines an expression to get a list of attribute definitions which are not loaded for each contact displayed in the team control to improve the display performance. Especially attributes with many values which are not relevant for the displayed information should be excluded.
- `disablecreate`
  Defines an expression to decide if the creation of new objects is not allowed in the team control.

Example

```
customize CPTeamControl <TeamRoom> {
  placeholdertext = expression { return #StrSearchEmailOrOrganisation; }
  noresulttext = expression { return #StrEnterValidEmail; }
  createtext = expression { return #StrInviteNewContact; }
  createhovertext = expression { return #AccordionCreateStr; }
  createsymbol = expression { return #SymbolUserNew; }
  createapplication = expression { return #RoomCreateSimpleListContact; }
  excludedattrdefs = { useracceptedusers, userrefusedusers, userknownusers }
  disablecreate = expression { return false; }
}
```

6.13 CPExpandTeamControl

The customization point `FSCTEAMROOM@1.1001:CPExpandTeamControl` is used to evaluate the additional information displayed for an entry when it's expanded in the team control. The customization can be changed for each object class depending on the room context.

Syntax

```
customize CPExpandTeamControl<objclass, roomobjclass> {
  expandedentry = expression {...}
  expandableinreducedmode = true;
}
```

Description:

- *objclass*
  The customization applies to objects of the defined object class.
- *roomobjclass*
  The object class of the room to which the current object is assigned to.
- expandedentry
  Defines an expression to get the additional information displayed on expanding an entry. The additional information must be returned as dictionary. The information contained in the dictionary is displayed in the expanded section. Note: If you want to display information for members of expanded groups, it is possible to include a list of sub dictionaries with detailed information for each group member.
- *expandableinreducedmode*
  Defines if entries of this object class are expandable in light dispatcher. In the xml dispatcher each entry is expandable by default.

Example

```
customize CPExpandTeamControl <User, TeamRoom> {
  expandedentry = expression {
    dictionary retdict = {};
    retdict.SetEntry(#StrEmail.Print(), cooobj.GetHoverText().email);
    retdict.SetEntry(#persorgtxt.Print(), cooobj.GetHoverText().orgtext);
    return retdict;
  }
  expandableinreducedmode = false;
}

customize CPExpandTeamControl <WorkGroup, TeamRoom> {
  expandedentry = expression {
    dictionary retdict = {};
    User[] members = cooobj.wgrmembers;
    coouser.Sort(&members, true, #objname);
    integer idx = 0;
    for (User member : members) {
      if (member && member.HasClass(#User)) {
        dictionary hoverdict = member.GetHoverText();
        dictionary entrydict = {
          value = :>member.GetName(),
          presymbol = #SymbolUser.GetAddress(),
          hover = :>hoverdict.hovertext,
          alignright = true,
          address = :>member.GetAddress()
        };
        idx++;
        string key = "" + string(idx) + ".";
        retdict.SetEntry(key, entrydict);
      }
    }
    return retdict;
  }
  expandableinreducedmode = true;
}
```

CPGetWBContainers

6.14 CPGETWBContainers

The customization point FSCTEAMROOM@1.1001:CPGETWBContainers is used to define which object classes should be considered as containers when calling FSCTEAMROOM@1.1001:GetOriginalChildren.

Syntax

```
customize CPGetWBContainers< sourceroomclass> {
  containerclasses = expression {...}
}
```

Description:

- *objclass*
  The customization applies to waste baskets of the defined object class.

  containerclasses
  Defines the object classes that should be considered as containers when calculating the original children of a room

Example

```
customize CPGetWBContainers <PersonnelFileShelf> {
  containerclasses = expression { return [#PersonnelFile, #ContentObject]; }
}
```

7 Transfer

7.1 AdditionalTransferObjects

The customization point `FSCTRANSFER@1.1001:AdditionalTransferObjects` calculates depending transfer objects which will be used by specific object classes. These depending objects will be transferred as copies and won't have a transfer state or a link between the object in the source and the target domain. This customization point is evaluated in the source domain when the container is transferred and in the partner domain when the container is retracted.

Syntax

```
customize AdditionalTransferObjects<objclass> {
  objects = expression {...}
}
```

Description:

- *objclass*
  The customization applies to objects of the defined object class.
- *objects*
  The depending objects for an instances of the object class.

In the expression following additional variables are available:

- *targetrootobj*
  The container object of the transfer. This parameter normally represents the team room which is transferred.
- *transferobj*
  The current object for which the depending objects should be calculated
- *transferdirection*
  The direction of the transfer. (e.g. sending or receiving)

Example

```
customize AdditionalTransferObjects <Mail> {
  objects = expression {
    cooobj.mailattacheditems;
```

```
  }
}
```

7.2 AdditionalTransferParticipants

The customization point `FSCTRANSFER@1.1001:AdditionalTransferParticipants` calculates additional transfer users which will be used by specific classes. By default only the users which have permissions on the transfer root object ware transferred.

Syntax

```
customize AdditionalTransferParticipants<objclass> {
  users = expression {...}
}
```

Description:

- *objclass*
  The customization applies to objects of the defined object class.
- *users*
  A list of users which should be transferred additionally to the participants of the transfer root.

In the expression following additional variables are available:

- *targetrootobj*
  The container object of the transfer. This parameter normally represents the team room which is transferred.
- *transferobj*
  The current object for which the additional participants should be calculated
- *transferdirection*
  The direction of the transfer. (e.g. sending or receiving)

Example

```
customize AdditionalTransferParticipants <Mail> {
  users = expression {
    cooobj.onlinerefs.user + cooobj.onlinerefsexclude.user;
  }
}
```

7.3 TransferContainer

The customization point `FSCTRANSFER@1.1001:TransferContainer` defines if an object can act as transfer container. By default a valid transfer container must have the attributes `FSCTRANSFER@1.1001:objtransferprotocol` and `FSCTRANSFER@1.1001:objtransferpublishprotocols` assigned.

Syntax

```
customize TransferContainer<objclass> {
  isvalidcontainer = expression {
  }
}
```

Description:

- *objclass*
  The customization applies to objects of the defined object class.

- *isvalidcontainer*
  The result defines if the current object can act as transfer container

Example

```
customize TransferContainer <Object> {
  isvalidcontainer = expression {
  return cooobj.CheckSetAccess(cootx, #objtransferstate) &&
         cooobj.HasAttribute(cootx, #objtransferprotocol) &&
         cooobj.HasAttribute(cootx, #objtransferpublishprotocols);
  }
}
```

7.4 TransferContainerLocation

The customization point `FSCTRANSFER@1.1001:TransferContainerLocation` defines the linking property between the current object and the transfer root. During the transfer the link between the two objects is created in the target domain.

Syntax

```
customize TransferContainerLocation<objclass, containerobjclass> {
  containerlocation = AttributeObjectDef
}
```

Description:

- *objclass*
  The customization applies to objects of the defined object class.
- *containerobjclass*
  The object class of the transfer root.
- *containerlocation*
  The property in which the link between the current object and the transfer root is stored.

Example

```
customize TransferContainerLocation <Object, TeamRoom> {
  containerlocation = objteamroom;
}
```

7.5 FinishTransfer

The customization point `FSCTRANSFER@1.1001:FinishTransfer` allows execute post steps for each object when it was transferred successfully. The configured expression must return true when post steps were executed correctly

Syntax

```
customize FinishTransfer<objclass> {
  steps = expression {...}
}
```

Description:

- *objclass*
  The customization applies to objects of the defined object class.

- *steps*
  The expression is executed directly after the object was transferred to perform post steps defined within. The expression must return if the steps have been executed correctly.

In the expression following additional variables are available:

- *targetrootobj*
  The container object of the transfer. This parameter normally represents the team room which is transferred.
- *transferobj*
  The current object for which the post steps should be executed
- *transferdirection*
  The direction of the transfer. (e.g. sending or receiving)

Example

```
customize FinishTransfer <Object, TeamRoom> {
  steps = expression {
    if (targetrootobj.HasClass(#Object) && transferobj.HasClass(#Object)) {
      TransferProtocol protocol = targetrootobj.GetTransferProtocol(false, false);
      if (protocol.HasClass(#TransferProtocol)) {
        transferobj.PatchRemarks(protocol.transfermapping);
      }
    }
    return true;
  }
}
```

7.6 ConcludeSecurity

The customization point `FSCTRANSFER@1.1001:ConcludeSecurity` defines which security is applied to the transferred objects.

Syntax

```
customize ConcludeSecurity<objclass> {
  secapplied = boolean
}
```

Description:

- *objclass*
  The customization applies to objects of the defined object class.
- *secapplied*
  The expression which defines the security for the object and returns if the security was applied

In the expression following additional variables are available:

- *targetrootobj*
  The container object of the transfer. This parameter normally represents the team room which is transferred.

Example

```
customize ConcludeSecurity <Object> {
  secapplied = expression {
    AccessControlList acl = cooobj.AttrObjACLConstructor(#objaclobj);
    cooobj.ObjectLock(true, true);
    cooobj.objaclobj = acl;
    return true;
  }
}
```

8 Personnel File

8.1 CPInstanceACLs

The customization point `CPInstanceACLs` defines the access control lists which can be used in the personnel file context. For example the possible acls for instances of personnel file folders are restricted to the defined values of the customization point.

Syntax

```
customize CPInstanceACLsQuickSearchAction<objclass> {
  acls = expression {...}
}
```

Description:

- *objclass*
  The customization applies to objects of the defined object class.
- *acls*
  The possible access control lists for objects in the personnel file context.

Example

```
customize CPInstanceACLsQuickSearchAction <PersonnelFileFolder> {
  acls = expression {
    return [#AppConfigurationRoomACL, , #RoomPersonnelFileShelfACL, #RoomPersonnelFileACL,
            #RoomPersonnelFileConfidentialACL, #RoomPersonnelFileSupervisorACL];
  }
}
```

8.2 InheritSecurityContext

The customization point `InheritSecurityContext` defines in which cases in an personnel file context the parent acl will be inherited by the child..

Syntax

```
customize InheritSecurityContext<objclass> {
  definedacl = expression {...}
}
```

Description:

- *objclass*
  The customization applies to objects of the defined object class.
- *definedacl*
  The acl which sould be applied to the child object. If no acl is returned the current acl of the child object will not be changed.

Example

```
customize InheritSecurityContext<Object> {
  definedacl = expression {
    AccessControlList curracl = cooobj.objaclobj;
    AccessControlList categoryacl = null;
    DocumentCategory category = cooobj.COOTC@1.1001:objcategory;
    if (category &&
        (category.HasClass(#DocumentCategory) ||
         category.HasClass(#ComponentDocumentCategory))) {
      categoryacl = category.dcdefaultacl;
    }
```

```
    if (categoryacl && categoryacl.HasClass(#AccessControlList)) {
      return null;
    }
    else if (cootx.IsCreated(cooobj) || cooobj in #TV.TV_ALLIMPORTEDOBJECTS ||
        cooobj in #TV.TV_PERSISTED || backlink != #objaclobj ||
        (curracl && curracl == oldparentacl)) {
      return newparentacl;
    }
  }
}
```

9 Search

9.1 CPQuickSearchAction

The customization point `CPQuickSearchAction` allows overriding the action to be called to perform the quick search in an object pointer property.

Syntax

```
customize CPQuickSearchAction<cfgobjclass, cfgview> {
  cfgqsaction = ...;
}
```

Description:

- *cfgobjclass*
  The customization applies to objects of the defined object class.
- *cfgview*
  The property, to which the customization applies when a quick search is executed.
- `cfgqsaction`
  Action to be called as quick search action.

The following example defines `SearchUser` as action for the quick search in the owner property.

Example

```
customize CPQuickSearchAction<Object, objowner> {
  cpqsaction = SearchUser;
}
```

9.2 CPQuickSearchAppearance

The customization point `CPQuickSearchAppearance` defines for which object class in which object pointer property the enhanced or simple appearance is used.

Syntax

```
customize CPQuickSearchAppearance<cfgobjclass, cfgattrdefopt> {
  appearance = ...;
}
```

Description:

- *cfgobjclass*
  The customization applies to objects of the defined object class.

- *cfgattrdefopt*
  The property, to which the customization applies when a quick search is executed.
- appearance
  Defines whether the results of the quick search in the property property are displayed “Enhanced” or “Simple”.

The following example defines the enhanced appearance for all objects, which are searched in the *Owner* (COOSYSTEM@1.1:objowner) property.

Example

```
customize CPQuickSearchAppearance<Object, objowner> {
  appearance = QS_ENHANCED;
}
```

9.3 CPQuickSearchSuffix

The customization point CPQuickSearchSuffix defines the properties to be displayed in the enhanced appearance. A search result of the object class cfgobjclass is displayed in the cfgattrdefopt list according to the additionaldescription and the cfgmultilinedescription expression. If cfgattrdefopt does not contain a value, the setting applies to all lists.

For the “Simple” appearance the result of the cfgmlnamesuffix expression is used. For the “Enhanced” appearance the result of the cfgmultilinedescription expression is used. If this expression is not defined, the cfgmlnamesuffix expression is used.

Syntax

```
customize CPQuickSearchSuffix<cfgobjclass, cfgattrdefopt > {
  cfgmlnamesuffix = expression {...}
  cfgmultilinedescription = expression {...}
}
```

Description:

- *cfgobjclass*
  The customization applies to objects of the defined object class.
- *cfgattrdefopt*
  Optionally the customization applies to this property.
- cfgmlnamesuffix
  This result can only be displayed in one line and is displayed after the object name.
- cfgmultilinedescription
  This Fabasoft app.ducx expression defines the additional properties to be displayed. The result can also be displayed in multiple lines. For displaying a multiline result \n is used for starting a new line.

The following example defines that for search results of the *User* (COOSYSTEM@1.1:User) object class also the default role is displayed.

Example

```
customize CPQuickSearchSuffix<User> {
   cfgmultilinedescription = expression {
     UserRoleList[] @defroles = cooobj.userroles[COOSYSTEM@1.1:default];
     string @rolename = "";
     if (count(@defroles) == 0) {
       @defroles = cooobj.userroles;
     }
```

```
      if (count(@defroles) > 0) {
        UserRoleList @defrole = @defroles[0];
        if (@defrole != null) {
          @rolename = @defrole.userrolepos.objname;
          @rolename += " "+@defrole.userrolegroup.grshortname;
        }
      }
      [cooobj.objname, @rolename];
    }
}
```

9.4 CPMindbreezeQueryConstraints

The customization point `CPMindbreezeQueryConstraints` provides a set of Mindbreeze constraints, which are used to restrict the search results of the Mindbreeze query to particular scopes.

Syntax

```
customize CPMindbreezeQueryConstraints<objclass, roomclass, constrainttype> {
  queryconstraintdicts = expression {...};
}
```

Description:

- *objclass*
  The customization applies to objects of the defined object class.
- *roomclass*
  The customization applies to objects of the defined room type.

  constrainttype
  Input Parameter to identify the context. It is an instance of the type `FSCVENV@1.1001:MindbreezeQueryConstraint`.
  `FSCVENV@1.1001:MQC_GENERAL` for example is used to calculate the default query constraints used in the default search of the browser client.
- `queryconstraintdicts`
  Defines a list of dictionaries that describe a search scope. The dictionaries consist of the following entries:
  - *id*
    The id of the query scope.
  - *name*
    The name of the query scope. This text will be shown in the autocomplete dropdown list of the search box.
  - *defaulttab*
    `true` if this scope should be the default scope, i.e. the scope that is used if the user does not choose particular scope from the autocomplete dropdown list of the search box.
  - *constraint*
    A Mindbreeze query constraint string that restricts the search to this scope.

The following example defines two scopes, one without a restriction and one within the current group.

Example

app.ducx Customization Language

```
customize CPMindbreezeQueryContraint<Folder, null, null> {
  queryconstraintdicts = expression {
```

```
      dictionary[] @constraints;
      if (cooobj.objowngroup) {
        Object @group = cooobj.objowngroup;
        @constraints = [
          {
            id: "group",
            name: "In Group \"" + @group.objname + "\"",
            defaulttab: true,
            constraint: "objowngroup/annotation/mes:key:" + @group.objaddress
          },
          {
            id: "all",
            name: "Everywhere",
            defaulttab: false,
            constraint: "ALL"
          }
        ];
      }
      return @constraints;
    }
}
```

10 Workflow

10.1 MetaParticipant

The customization point `MetaParticipant` defines how the meta participant is resolved.

Syntax

```
customize MetaParticipant<objclass, metapart> {
  newparticipant = expression {...}
}
```

Description:

- *objclass*
  The customization applies to objects of the defined object class.
- *metapart*
  Defines the meta participant that should be resolved.
- newparticipant
  Defines the expression that is used to resolve the meta participant to a participant.

In the expression following additional variables are available:

- *object*
  The object, on which the process is running.
- *activity*
  The current activity instance.
- *process*
  The object of the current process.
- *participant*
  The current participant of the activity.

Example

```
customize MetaParticipant<Object, WFMP_INITIATOR> {
  newparticipant = expression {
    @owner = object.objowner;
    COOWF@1.1:WorkFlowParticipant @part = {};
```

```
    if (@owner != null) {
      @roles = @owner.userroles[COOSYSTEM@1.1:default==true];
      if (count(@roles) > 0) {
        @role = @roles[0];
        @rolegroup = @role.userrolegroup;
        if (@rolegroup != null) {
          if (@role.userrolepos == #FSCFOLIO@1.1001:HeadPos) {
            @supergroups = @rolegroup.grsupergroups;
            if (count(@supergroups) > 0) {
              @part.COOWF@1.1:wfpgroup = @supergroups[0];
            }
            else {
              @part.COOWF@1.1:wfpgroup = @rolegroup;
            }
            @part.COOWF@1.1:wfpposition = #FSCFOLIO@1.1001:HeadPos;
          }
          else {
            @part.COOWF@1.1:wfpgroup = @rolegroup;
            @part.COOWF@1.1:wfpposition = #FSCFOLIO@1.1001:HeadPos;
          }
        }
        else {
          @part.COOWF@1.1:wfpmetaparticipant =
            participant.COOWF@1.1:wfpmetaparticipant;
        }
      }
      else {
        @part.COOWF@1.1:wfpmetaparticipant =
          participant.COOWF@1.1:wfpmetaparticipant;
      }
    }
    else {
      @part.COOWF@1.1:wfpmetaparticipant =
        participant.COOWF@1.1:wfpmetaparticipant;
    }
    @part;
  }
}
```

Note: In the Fabasoft Folio Domain the customization gets defined in the workflow configuration.

10.2 ProcessOwnership

The customization point `ProcessOwnership` defines the owner and the group of the workflow process. If no configuration was made the current user is used as the owner of the process and the group of the current role will be used as owning group of the process.

The customization point `ProcessOwnership` is evaluated when creating a new process in the action `COOWF@1.1:CreateWorkFlowObject`.

Syntax

```
customize ProcessOwnership<objclass> {
  owner = expression {…}
  group = expression {…}
}
```

Description:

- *objclass*
  The customization applies to objects of the defined object class.
- *owner*
  Defines an expression to get the owner of the process.

- *group*
  Defines an expression to get the owning group of the process.

Example

app.ducx Customization Language

```
customize ProcessOwnership<Object> {
  owner = expression {
    TeamRoom tr = cooobj.objteamroom;
    if (tr && tr.HasClass(#TeamRoom)) {
      return tr.objowner;
    }
    else {
      return cooobj.objowner;
    }
  }
  group = expression {
    TeamRoom tr = cooobj.objteamroom;
    if (tr && tr.HasClass(#TeamRoom)) {
      return tr.objowngroup;
    }
    else {
      return cooobj.objowngroup;
    }
  }
}
```

10.3 InsertActivityDef

The customization point `InsertActivityDef` calculates an activity definition for a specified object class with a supplied context.

Syntax

```
customize InsertActivityDef<objclass, scope> {
  insertactdef = expression {...}
}
```

Description:

- *objclass*
  The customization applies to objects of the defined object class.
- *scope*
  A component object, that is used in combination with the object class to retrieve an activity definition.
- *insertactdef*
  Defines a Fabasoft app.ducx Expression to calculate the activity definition. Following values are available in the local scope:
  - activity: The currently used activity instance of the current process
  - process: The currently used process instance
  - participant: The participant of the current activity instance
  - object: The object of the current process

Example

app.ducx Customization Language

```
customize InsertActivityDef<Object, SIGN_APPROVAL_ACCEPTED> {
```

```
  insertactdef = expression {
    return #ActDefApprovalAccepted;
  }
}
```

Note: In the Fabasoft Folio Domain this customization gets defined in the workflow configuration.