Project information

Invite your team

Add members to this project and start collaborating with your team.

Created on
December 14, 2022

Embedded Mailer

Der Embedded Mailer für enaio® ist eine Erweiterung, mit der sich leicht E-Mails adressieren, schreiben und mit Anhängen versehen lassen und direkt aus einem enaio®-Client heraus versendet werden können.

Inhaltsverzeichnis

- Embedded Mailer
  - Inhaltsverzeichnis
  - Voraussetzungen
- Installation
  - Voraussetzungen
  - apps-Verzeichnis
  - servicewatch-sw.yml
  - dashlets.json
  - embedder-mailer.yaml
  - service-manager
- Benutzerhandbuch
  - Auswahl von E-Mail-Adressen
    - Adressen
      - Absender
    - Option: Kopie an Absender
      - Empfänger
      - Kopie an
      - Blindkopie an
    - Administration
  - Betreff und Nachricht
    - Betreff
    - Nachricht
    - Option: Ohne Signatur
    - Option: Nur Text
    - Administration
  - Anhänge
    - Anhänge auswählen
    - Anhänge (zur E-Mail) hinzufügen
      - Anhänge umbennen
    - Option: Interne E-Mail
    - Option: Möglichst als PDF
    - Administration
  - Nachrichtenvorlagen
    - Nachrichtenvorlage auswählen
    - Nachrichtenvorlage Konfigurieren
      - Platzhalter
    - Administration
- Administration
  - Komponenten des Embedded Mailer
  - Layout für E-Mails (CorporateDesign)
    - E-Mail Layouts
    - Layouts verwalten
      - Beispiel: layout-preview.html
    - Platzhalter innerhalb von Layouts
      - Platzhalter sind Context-Abhängig
      - Spezielle Objekte
        - data
        - data.attachments

        - Obligatorische, allgemeine Funktionen
    - getEmails()
    - getFileName()
    - getObjectLocation()
    - getPlaceholder()
    - persistEmail()
    - getTemplates()
- Optionale Erweiterungen
    - Konfiguration und Freigabe
        - Erweiterung: allowSignature

Voraussetzungen

- enaio® Version 9 oder höher
- enaio® WebClient
- enaio® Benutzer-Account mit gültiger E-Mail-Adresse
- Gültiger ECMind GmbH Lizenzschlüssel

Installation

Voraussetzungen

- Embedded Mailer Erweiterung als .jar-Datei (ggf. in Archiv (z.B. .zip) geliefert)
- Betriebsbereite enaio® Installation mit Zugriff auf service-manager -Verzeichnis
- Betriebsbereiter SMTP-Server zum versenden von E-Mails
- Firewall lässt Requests & Responses von Port 9353 immer zu

apps-Verzeichnis

Im Verzeichnis `Pfad\zu\Ihren\OptimalSystemsKomponenten\services\service-manager\apps` ein neues Unterverzeichnis `embedded-mailer` anlegen.

In dieses Verzeichnis muss einmalig für die Erstinstallation die Datei `mailing-app.jar` aus dem Embedded Mailer ZIP-Archiv hinterlegt werden.

servicewatch-sw.yml

In der Datei `Pfad\zu\Ihren\OptimalSystemsKomponenten\services\service-manager\config\servicewatcher-sw.yml` folgenden Abschnitt ganz unten hinzufügen:

```
- name: embedded-mailer
  type: microservice
  profiles: prod,cloud
  instances: 1
  memory: 256M
  port: 9353
  arch: x64
  path: ${appBase}/embedded-mailer/mailing-app.jar
```

dashlets.json

In der Datei `Pfad\zu\Ihren\OptimalSystemsKomponenten\services\service-manager\config\apps\osweb\public\dashlets\dashlets.json` folgenden Abschnitt ganz unten hinzufügen (ohne eckige Klammer am Anfang und Ende) oder oben einfügen:

```
[
    {
        "objectTypes": "*",
        "platforms": ["web", "desktop_app", "mobile", "mobile_app"],
        "uri": "http://[ENAIO-BASE-URI]/mailing/",
        "title_EN": "Embedded Mailer",
        "title_DE": "Embedded Mailer",
        "title_FR": "Embedded Mailer",
        "iconId": "1"
```

embedder-mailer.yaml

Im Verzeichnis `Pfad\zu\Ihren\OptimalSystemsKomponenten\services\service-manager\config\` die Datei `embedded-mailer-prod.yaml` anlegen und folgende Konfiguration einfügen. Anschliessend alle nötigen Angaben auf das System anpassen.

```
mailing:
    backend:
        client: enaio
        licenseKey: YOUR-COMPANIES-LICENSE-KEY
        licenseVersion: 2
    paths:
        - type: mailing-dashlet-design
          visibility: private
          target: Pfad\zu\Ihren\OptimalSystemsKomponenten\services\service-manager\data\embedded-mailer\designs\your
    attachments:
        # 3 Types: EmptyJob | MandatoryFields | SystemFields
        fileNameGenerator:
            type: MandatoryFields
        # @todo Systemfields Konfiguration in MetadataService einbauen
        systemFields:
            - objectId
            - objectTypeId
            - lastModificationDate
# Defines, which contact-types (currently AppConnector endpoint) getContacts should consider
contacts:
    anonymous: false
    contacts: true
    user: true
    internal: true
    limit: 50
extensions:
    # optional extensions
    frontend:
        - name: allowSignature
          type: EmptyJob
          parameters:
          expect: Boolean
    backend:
        # - name: closureTest
        #   type: Closure
        #   expect: List
        #   parameters:
        #   param1: 1
        #   param2: test
        #   param3: [1, 2, blubb]
eml:
    store: true
endpoints:
    appConnector:
        url: http://[ENAIO-BASE-URI]:8060/osrest/api
        actions:
            - name: callEmptyJob
              query: ?timeout=120000
    renditionCache:
        url: http://[ENAIO-BASE-URI]:8070/osrenditioncache
        actions:
            - name: convertToPDF
              query: ?timeout=120000
    smtp:
        host: your-mail-smtp-host
        port: 0
        enable-ssl: true
        enable-startTls: true
```

```
            email: your-default-email@your-company-domain.ch
            password: *******
        anonymous:
            enable: true
            email: anonymous@your-company-domain.ch # non-personal email dispatching
```

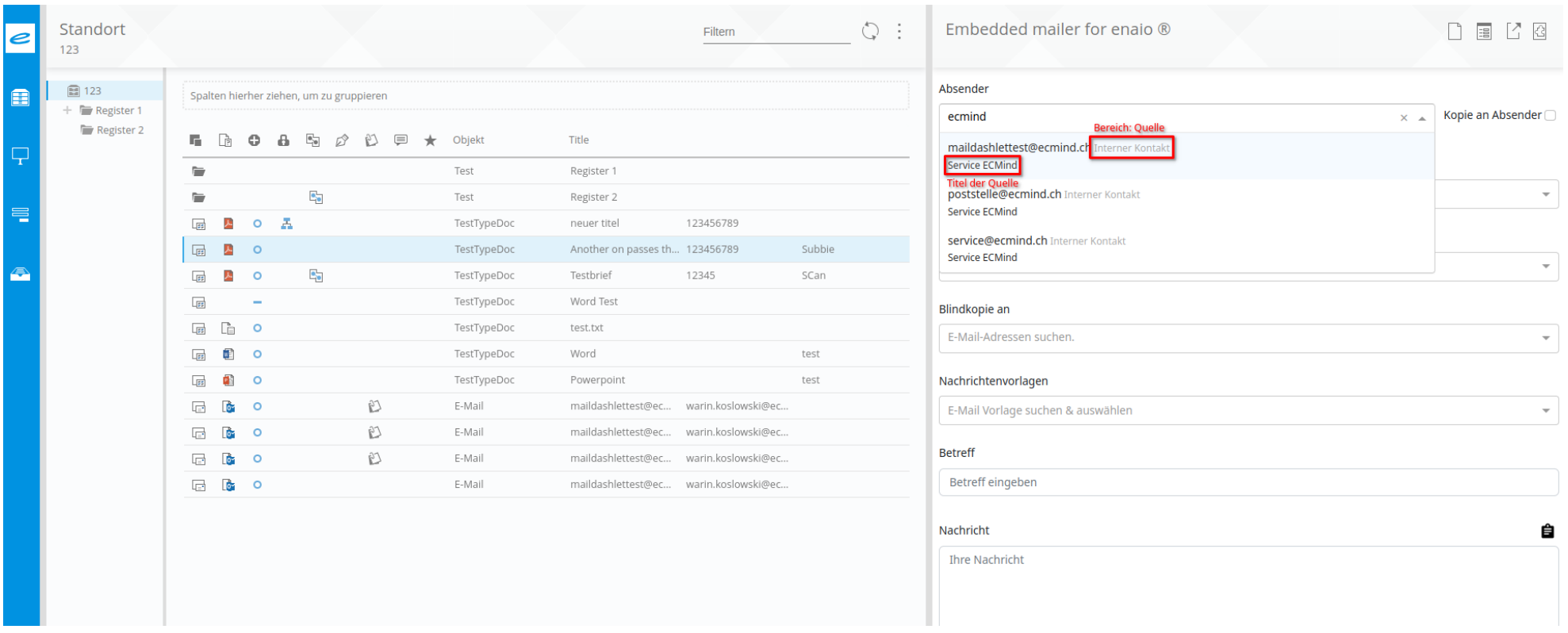
service-manager

Sind alle Änderungen gemacht, muss der enaio® service-manager neu gestartet werden.

Nach oben

Benutzerhandbuch

Auswahl von E-Mail-Adressen



Adressen

Folgende E-Mail-Adressen können ausgewählt werden:

Absender

Mit dieser Adresse wird festgelegt, wer diese E-Mail verschickt. Sofern vorhanden, kann dieses Feld mit der E-Mai-Adresse des Benutzers und auch anderen, vom Administrator definierten, E-Mai-Adressen vorbelegt werden.

Option: Kopie an Absender

Versendet eine Kopie der E-Mail an die E-Mail-Adresse des ausgewählten Absenders.

Empfänger

An eine oder mehrere E-Mail-Adressen, die hier ausgewählt wurden, wird die Nachricht verschickt. In den meisten E-Mail-Clients kann hier auch problemlos eine Antwort an alle hier verwendeten Adressaten verschickt werden.

Kopie an

Werden hier ein oder mehrere E-Mail-Adressen ausgewählt, so erhalten diese eine Kopie der E-Mail - werden jedoch nicht als Hauptadressaten geführt.

Blindkopie an

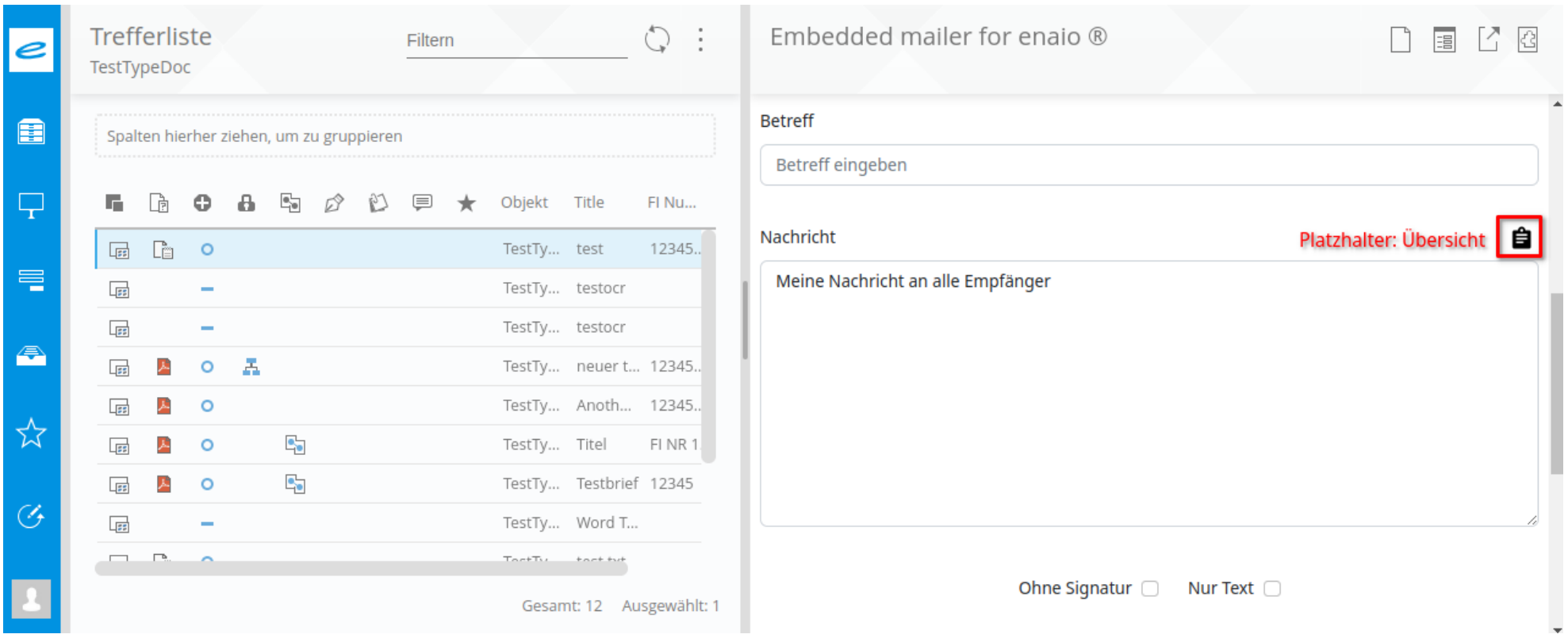
Eine Blindkopie ist eine "versteckte" Kopie an einen odere mehrere Adressaten. Diese sind nicht als Empfänger zu erkennen und sind bei den sonstigen Adressaten nicht ersichtlich.

Administration

Bereich & Titel der Quelle können in enaio® VBS `EmptyJob` in Funktion `getEmails()` definiert werden.

Hierbei bezeichnet der Bereich normalerweise den Ordners und der Titel ist der konkrete Name eines Typs.

Betreff und Nachricht



Betreff

Der Betreff ist ein einfaches Textfeld. Formatierungen sind hier nicht möglich und es gibt ein natürliches Limit seitens enaio® oder auch von den meisten E-Mail-Clients.

Nachricht

Eine E-Mail Nachricht kann innerhalb des Embedded Mailers mittels MarkDown Basic-Syntax formatiert werden:

Markdown-Guide

Darüber hinaus lassen sich auch Platzhalter im Nachrichtentext verwenden. Eine Liste der verfügbaren Platzhalter kann jederzeit durch einen Click auf das Klemmbrett oberhalb der Nachrichtenbox aufgerufen werden. Alle verfügbaren Platzhalter können in der Syntax `${PLATZHALTERNAME}` verwendet werden.

Option: Ohne Signatur

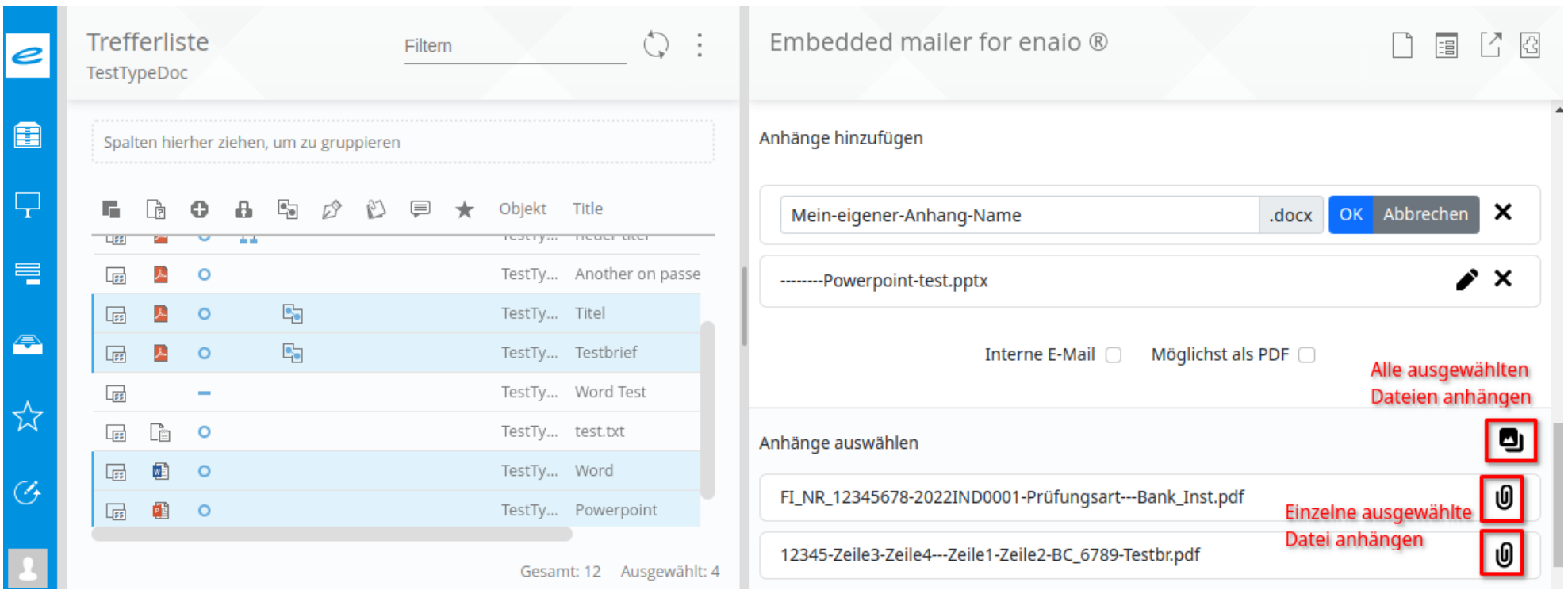
Die Option "Ohne Signatur" kann verwendet werden, um E-Mails ohne - ansonsten automatisch angehängter - Signatur zu versenden. Das Corporate Design, das von einem Administrator konfiguriert werden kann, wird weiterhin verwendet.

Option: Nur Text

"Nur Text" ist ebenfalls optional und blendet neben einer Signatur auch jegliches Corporate Design aus. Hiermit können völlig "neutrale" E-Mails verschickt werden, die keinen Bezug zum Unternehmen mehr haben.

Administration

Platzhalter, die innerhalb von E-Mail-Nachrichtentexten eingesetzt werden dürfen, können in enaio® VBS `EmptyJob` in Funktion `getPlaceholder()` individuell und frei definiert werden.



Anhänge auswählen

Durch ein Anklicken eines oder Auswahl mehrerer Einträge in der enaio® Trefferliste oder einer Standort-Ansicht, werden die markierten Dokumente zur Auswahl als E-Mail-Anhang im Bereich "Anhänge auswählen" bereit gestellt und mit einem vom System ermittelten Dateinamen versehen.

Anhänge (zur E-Mail) hinzufügen

Ein Anklicken der Büroklammer neben dem Dateinamen in der Liste "Anhänge auswählen" wird ein Anhang tatsächlich zur E-Mail hinzugefügt und mit der E-Mail-Nachricht an den Empfänger versendet.

Anhänge umbennen

Anhänge werden zunächst mit einem vom System generierten Dateinamen vorgeschlagen. Durch einen Doppelklick auf den Namen oder durch das Klicken auf das Bleistift-Symbol neben dem Dateinamen, kann der Name bearbeitet werden. Wird die Änderung mit OK bestätigt, wird der Anhang unter diesem Namen versandt.

Option: Interne E-Mail

Wird eine E-Mail als interne Nachricht versendet, gibt es die Anhänge nicht mehr als physische Datei an der E-Mail selbst. Stattdessen werden enaio® interne Links angeboten, die es nur in enaio® eingeloggten Benutzern erlauben, die Dateien zu sehen.

Option: Möglichst als PDF

Ist die Option "Möglichst als PDF" gesetzt (schliesst "Interne E-Mail" aus), wird versucht aus allen ausgewählten Anhängen ein PDF zu machen. Mit den meisten Windows® Dokument-Formaten ist dies möglich. XML-Dateien oder auch Bildformate können nicht in PDFs verwandelt werden und werden im Original-format versendet.

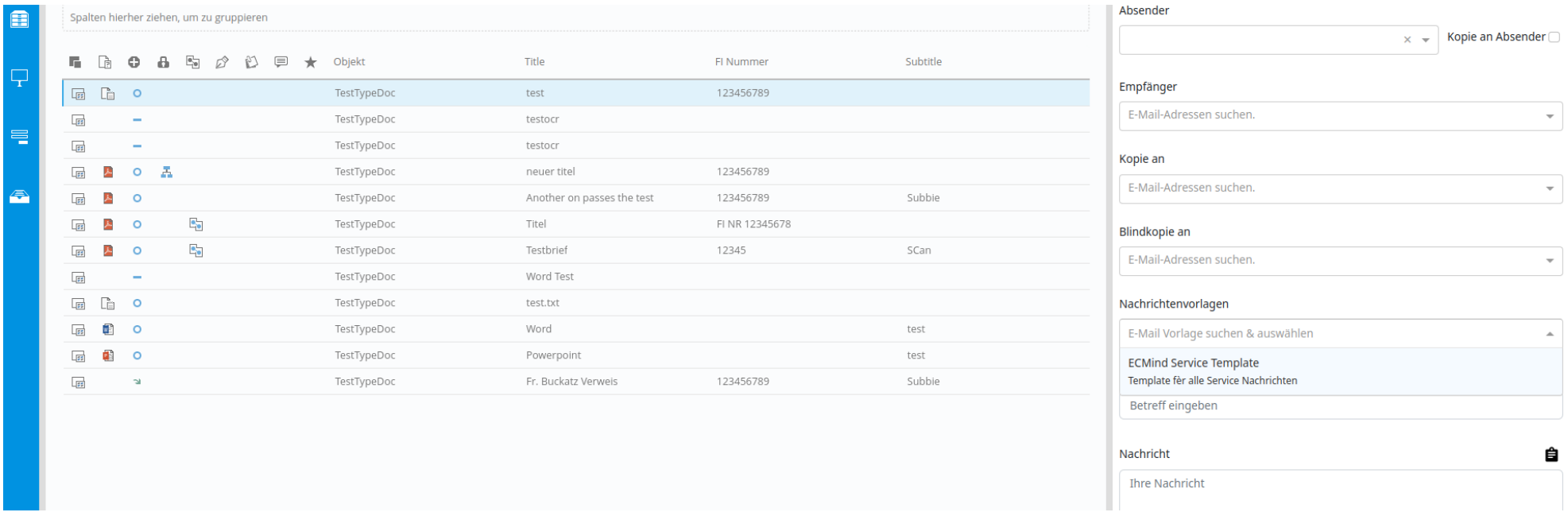
Administration

Der Name eines Anhangs wird zunächst vom System vorgeschlagen. Dieser Namensvorschlag kann von einem Administrator in enaio® VBS `EmptyJob` in Funktion `getFileName()` nach eigenen Regeln gemacht und lässt sich individuell anpassen.

Nachrichtenvorlagen

Nachrichtenvorlage auswählen

Durch ein Anklicken eines oder Auswahl mehrerer Einträge in der enaio® Trefferliste oder einer Standort-Ansicht, werden die markierten Dokumente zur Auswahl als E-Mail-Anhang im Bereich "Anhänge auswählen" bereit gestellt und mit einem vom System ermittelten Dateinamen versehen.



Nachrichtenvorlage Konfigurieren

Nachrichtenvorlagen können bei Bedarf via enaio® Richclient wie auch enaio®WebClient verwaltet werden, sofern entsprechende Berechtigungen dazu vorhanden sind. Abgesehen von den Feldern Name(Auswahl), Beschreibung, Betreff und E-Mail-Text, können individuell weitere Felder im enaio® Editor für Nachrichtenvorlagen konfiguriert werden.

Platzhalter

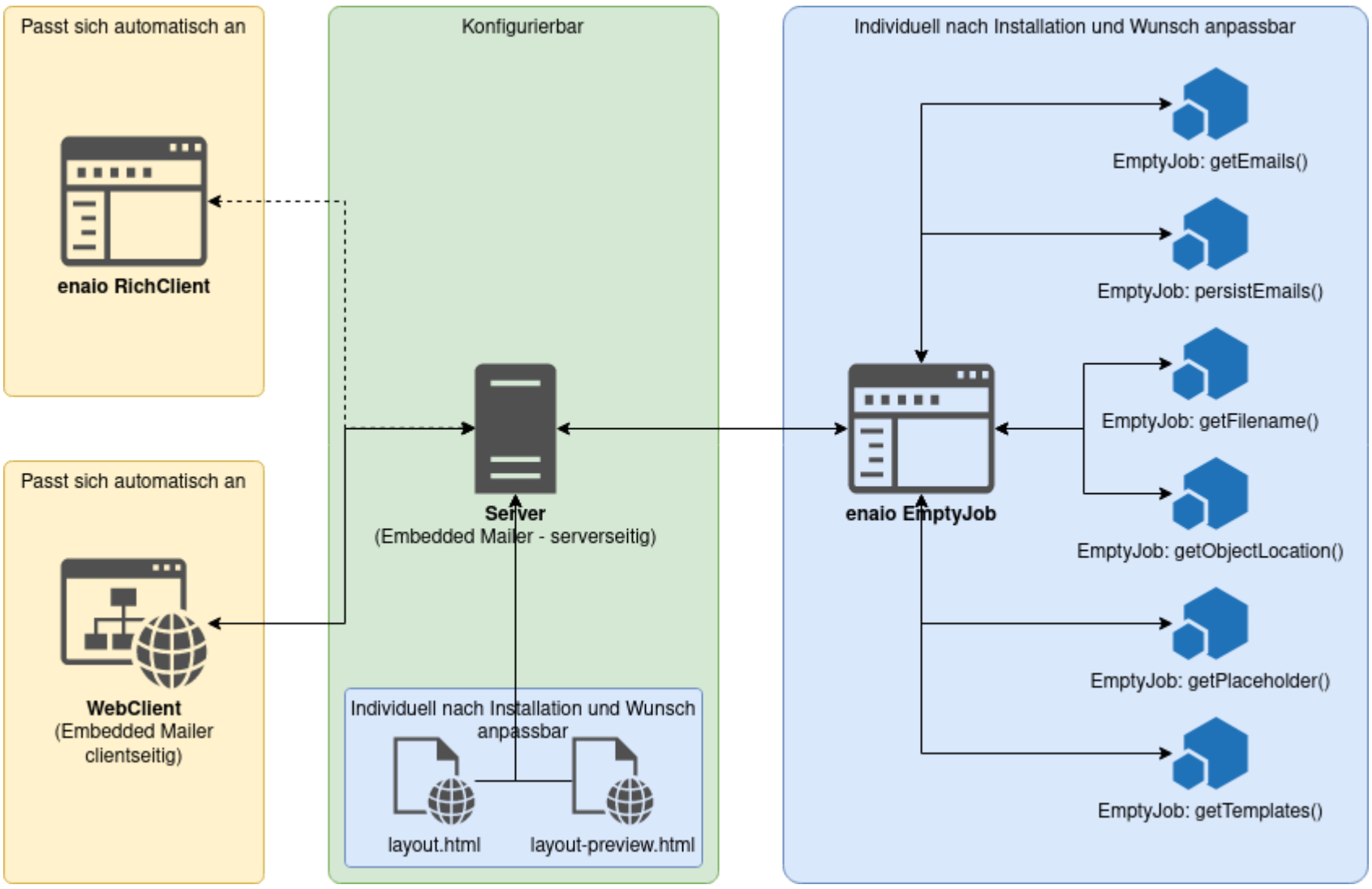
Wie im Nachrichtentext des Embedded Mailers selbst können im Nachrichtenvorlangen E-Mail-Text die selben Platzhalter verwendet werden.

Administration

Nachrichtenvorlagen können von einem Administrator in enaio® VBS `EmptyJob` in Funktion `getTemplates()` nach eigenen Regeln aufgearbeitet und in der Auswahlbox des Embedded Mailers verfügbar gemacht werden. Sie lassen sich individuell anpassen. Dies ermöglicht eine vielfältige Steuerung bzgl. der Nachrichtenvorlangen.

Nach oben

Administration



Layout für E-Mails (CorporateDesign)

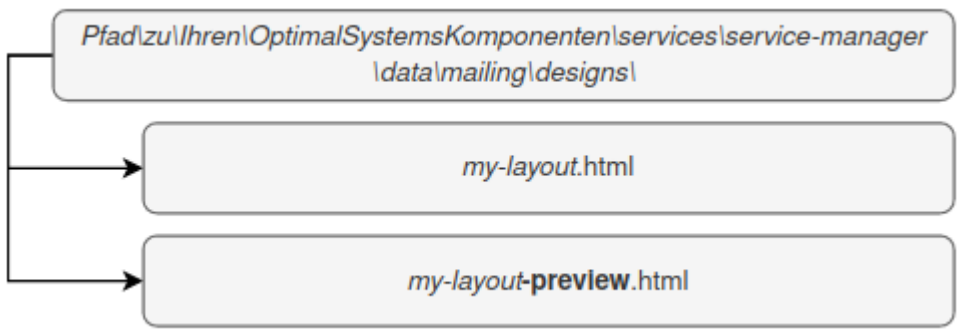
E-Mail Layouts

E-Mail-Layouts bieten die Möglichkeit, das Erscheinungsbild der vom Embedded Mailer versendeten E-Mails individuell zu gestalten um beispielsweise das firmeneigene CorporateDesign zu berücksichtigen.

Layouts verwalten

Layouts sind für E-Mails optimierte HTML-Dateien die in einem konfigurierbaren Verzeichnis abgelegt sind. Standardmässig ist dieses Verzeichnis `Pfad\zu\Ihren\OptimalSystemsKomponenten\services\service-manager\data\mailing\designs`. Vorschau-HTML für Vorschau im Embedded Mailer

Neben dem eigentlichen Layout muss auch ein Vorschau-Layout angelegt werden. Es trägt den selben Name wie das Layout selbst mit dem Unterschied, dass der Dateiname noch einen zusätzlichen postfix: preview enthält.



Beispiel: layout-preview.html

```
<div class="preview">
    <div>${parsedMessage}</div>

   <#if attachmentsAsLinks>
       <ul>
          <#list data.attachments as item>
             <li>
                <a href="${item.getUri()}">${item.getName()}</a>
             </li>
          </#list>
       </ul>
    </#if>
</div>
```

Wie im Embeded Mailer lassen sich auch im Layout alle verfügbaren Platzhalter verwenden. Die Notation bzw. Syntax ist ebenfalls gleich `${PLATZHALTERNAME}`.

Platzhalter sind Context-Abhängig

Platzhalter werden über den enaio® VBS `EmptyJob` in Funktion `getPlaceholder()` nach individuell beschlossenen Regeln aufgearbeitet. Es sollte sichergestellt sein, dass im Layout immer alle Platzhalter verfügbar sind.

Spezielle Objekte

data

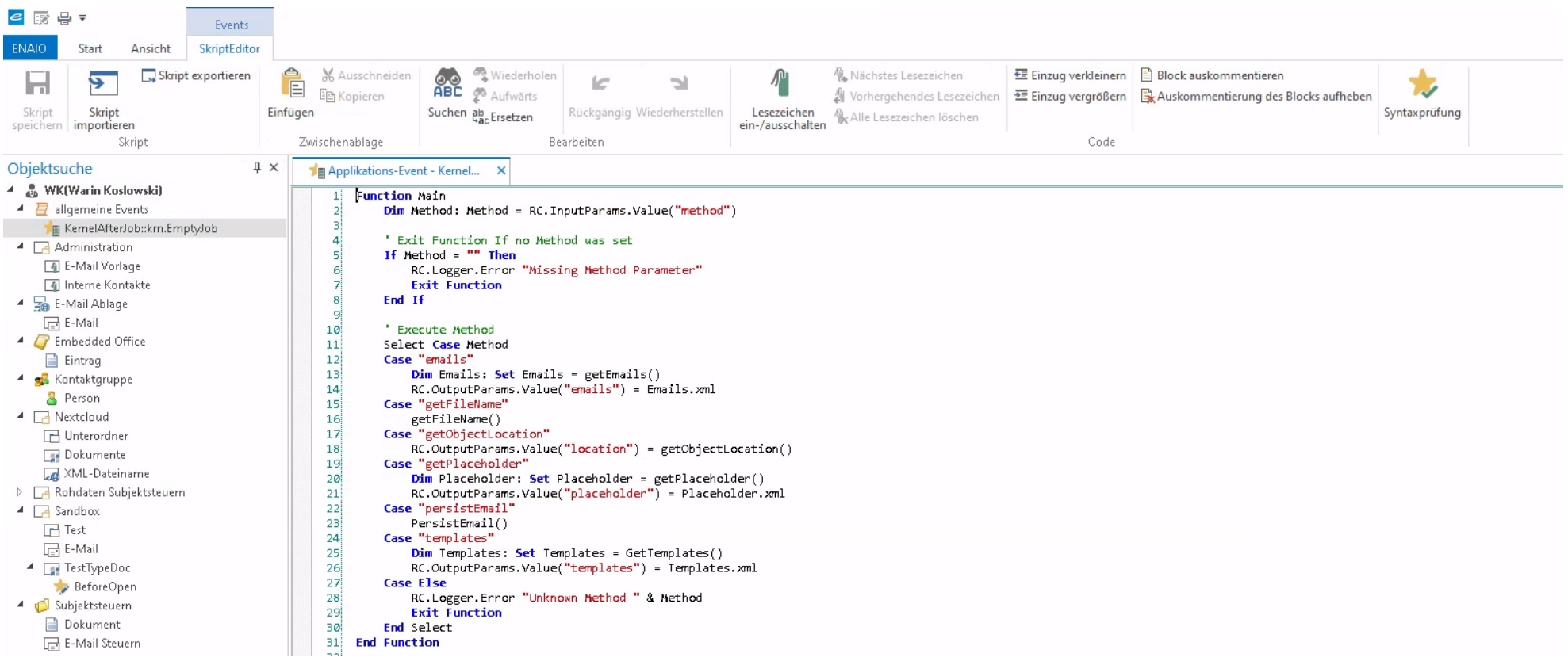
Data ist ein globales Objekt, das alle nutzbaren Platzhalter sowie spezielle Daten wie z.B. eine Liste der Anhänge, die an die E-Mail angehängt werden sollen.

data.attachments

`data.attachments` ist eine Liste mit den ausgewählten Anhängen einer E-Mail.

Nach oben

EmptyJob-Event in enaio®



EmptyJob

Der krnEmptyJob ist ein spezielles eanio® Event, das es erlaubt, in VBScript bestimmte Bedingungen und Funktionen zu definieren.

Obligatorische, allgemeine Funktionen

Diese Funktionen sollten immer im EmptyJob vorhanden sein.

```
' Hauptfunktion die auf HTTP-Anfragen des EmptyJobs reagiert.
Function Main
        Dim Method: Method = RC.InputParams.Value("method")

        ' Exit Function If no Method was set
        If Method = "" Then
                RC.Logger.Error "Missing Method Parameter"
                Exit Function
        End If

        ' Endpunkt / Funktion die an Hand der HTTP-Anfrage ausgeführt werden soll
    ' Grundregel:
    ' Alle für Auswahlboxen (DropDowns) verwendeten Funktionen müssen eine XML-Antwort zurückliefern
    ' Alle anderen Funktionen haben unterschiedliche Rückgabeformate (Text oder JSON)
        Select Case Method
        Case "emails"
            Dim Emails: Set Emails = getEmails()
```

```
            getFileName()
        Case "getObjectLocation"
            RC.OutputParams.Value("location") = getObjectLocation()
        Case "getPlaceholder"
            Dim Placeholder: Set Placeholder = getPlaceholder()
            RC.OutputParams.Value("placeholder") = Placeholder.xml
        Case "persistEmail"
            PersistEmail()
        Case "templates"
            Dim Templates: Set Templates = getTemplates()
            RC.OutputParams.Value("templates") = Templates.xml
        Case Else
            RC.Logger.Error "Unknown Method " & Method
            Exit Function
        End Select
End Function


' Funktion zur Dekodierung eines base64-kodierten Strings
Function Base64Decode(ByVal vCode)
    Dim oXML, oNode

    Set oXML = CreateObject("Msxml2.DOMDocument.3.0")
    Set oNode = oXML.CreateElement("base64")

    oNode.dataType = "bin.base64"
    oNode.text = vCode

    'Wichtig: enaio: us-ascii Kodierung, enaio unicode: UTF-16 Kodierung
        Base64Decode = Stream_BinaryToString(oNode.nodeTypedValue, 'UTF-16')

        Set oNode = Nothing
    Set oXML = Nothing
End Function

' Funktion zur Konvertierung eines binär-strings in eine per CharSet definierte andere Kodierung
Function Stream_BinaryToString(Binary, CharSet)
  Const adTypeText = 2
  Const adTypeBinary = 1

  Dim BinaryStream: Set BinaryStream = CreateObject("ADODB.Stream")

  BinaryStream.Type = adTypeBinary
  BinaryStream.Open
  BinaryStream.Write Binary
  BinaryStream.Position = 0
  BinaryStream.Type = adTypeText
  BinaryStream.CharSet = CharSet

  Stream_BinaryToString = BinaryStream.ReadText

  Set BinaryStream = Nothing
End Function

' Auswertung und Vorbelegung des HTTP-Parameters 'maxResults' der bei den meisten Anfragen vom Dashlet zu enaio mitg
' Der Parameter kan dazu verwendet werden, Ergebnismengen zu limitieren um Performance und Last zu verbessern.
Function getMaxResults()
        Dim maxResults: maxResults = RC.InputParams.Value("maxResults")
    ' Sofern Parameter nicht oder leer geschickt wurde kann hier der Standard (z.B. 100) festgelegt werden.
        If maxResults = "" OR maxResults = 0 Then
                maxResults = 100
        End If

        getMaxResults = maxResults
End Function
```

```
Function AdoSelect(ByVal Command)
                Dim JobInputParameter: Set JobInputParameter = RC.NewJobsParams
                Dim JobOutputParameter: Set JobOutputParameter = RC.NewJobsParams

                With JobInputParameter
                        .Value("Flags") = 1
                        .Value("Command") = Command
                End With

                RC.Jobs.ado.ExecuteSQL jobInputParameter, jobOutputParameter

                Dim RS: Set RS=CreateObject("ADODB.Recordset")

                RS.Open(JobOutputParameter.Value("ResultSet"))

                Set AdoSelect = RS
End Function

' Klasse um XML-Elemente zu generieren. Sie wird benötigt, um für jede Anfrage eine XML-Antwort zu generieren.
Class XMLElementClass
        Private MyDom
        Private MyElement

        Public Function Init(ByRef Dom, ByRef Element)
                Set MyDom = Dom
                Set MyElement = Element
        End Function

        Public Function Child(byVal TagName)
                Dim ChildElement: Set ChildElement = MyDom.createElement(TagName)
                MyElement.AppendChild ChildElement

                Dim ChildObj: Set ChildObj = New XMLElementClass
                ChildObj.Init MyDom, ChildElement
                Set Child = ChildObj
        End Function

        Public Function ChildWithContent(byVal TagName, ByVal Content)
                Dim ChildObj: Set ChildObj = Child(TagName)
                ChildObj.Content(Content)
                Set ChildWithContent = ChildObj
        End Function

        Public Function Attr(byVal Name, ByVal Value)
                MyElement.setAttribute Name, Value
        End Function

        Public Function Content(ByVal Value)
                MyElement.AppendChild MyDom.CreateTextNode(Value)
        End Function

        Public Function ToString
                ToString = MyDom.xml
        End Function

        Public Function ToDom
                Set ToDom = MyDom
        End Function
End Class

' Hilfsfunktion um die Verwendung der Klasse XMLElementClass zu vereinfachen
Function XMLElement(ByVal TagName)
        Dim MyDom: Set MyDom = CreateObject("MSXML2.DOMDocument")
        MyDom.documentElement = MyDom.createElement(TagName)
```

```
        Set XMLElement = ElementObj
End Function
```

Nach oben

getEmails()

Die Funktion holt alle E-Mail-Adressen, die im Embedded Mailer zur Auswahl für den Benutzer stehen können.

```
' MINIMAL VERSION
Function getEmails()
    ' E-Mail-Objekt anlegen (zum einfachereren erstellen einer XML-Antwort an das Dashlet)
    Set Email = CreateObject("Scripting.Dictionary")
    Email.Add "name" , "Max Mustermann"
    Email.Add "email" , "max@mustermann.ch"
    Email.Add "source", "System"

    ReDim Preserve Emails(UBound(Emails)+1)
    Set Emails(UBound(Emails)) = Email

    Dim count: count = 0
        Dim Response: Set Response = XMLElement("emails")

    ' Abschliessende Generierung der XML-Antwort an das Dashlet
        With Response
                .Attr "size", UBound(Emails)+1
                For Each Email In Emails
                        With .Child("email")
                                .Attr "name", Email("name")
                                .Attr "email", Email("email")
                                .Attr "source", Email("source")
                        End With
                        count = count + 1
                Next
        End With

        Set GetEmails = Response.toDom
End Function
```

```
Function getEmails()
    ' Anonyme Email-Adressen laden -> Werte: 1 oder 0
      Dim anonym: anonym = RC.InputParams.Value("anonymous")
' Email-Adressen von Benutzern laden -> Werte: 1 oder 0
      Dim mailsUser: mailsUser = RC.InputParams.Value("emailsUser")
' Email-Adressen von Kontakten (enaio-Ordner) laden -> Werte: 1 oder 0
      Dim mailsContacts: mailsContacts = RC.InputParams.Value("emailsContacts")
' Email-Adressen von internen Kontakten (enaio-Ordner) laden -> Werte: 1 oder 0
      Dim mailsInternal: mailsInternal = RC.InputParams.Value("emailsInternal")
' Maximale Anzahl der Ergebnisse -> Wert: Ganzzahl (Integer)
      Dim MaxResults: MaxResults = getMaxResults()
' Suchbegriff für die Suche nach passenden E-Mail-Adressen oder Benutzern in enaio -> Wert: text (String)
      Dim search: search = RC.InputParams.Value("searchQuery")

      Dim Emails: Emails = Array()
      Dim EmailsNodes, EmailNode, Email, FullName, Result, source, emailAdress

' Feste E-Mail-Adressen

' source legt die Quelle (Ansicht in Auswahlbox) der E-Mail fest.
source = "System"
' E-Mail-Objekt anlegen (zum einfachereren erstellen einer XML-Antwort an das Dashlet)
Set Email = CreateObject("Scripting.Dictionary")
```

```
    Email.Add "source", source

    ReDim Preserve Emails(UBound(Emails)+1)
    Set Emails(UBound(Emails)) = Email

    ' Ein weiteres E-Mail-Objekt anlegen
    Set Email = CreateObject("Scripting.Dictionary")
    Email.Add "name" , "Martha Mustermann"
    Email.Add "email" , "martha@mustermann.ch"
    Email.Add "source", source

    ReDim Preserve Emails(UBound(Emails)+1)
    Set Emails(UBound(Emails)) = Email

    ' E-Mail-Adresse eines oder mehrerer bestimmter Benutzer
    ' Flag emailsUser/mailsUser das via HTTP-Request von Dashlet zu enaio geschickt wird belegt die Variable mailsUs
        If mailsUser = 1 Then
                source = "Benutzer"
        ' Datenbankanfrage via ADODB, um Benutzerdaten zu laden
                Dim query: query = "SELECT * FROM benutzer WHERE flags=1 AND osemail <> ''"

        If Not search = "*" Then
                query = query & " AND (name LIKE '" & search & "%' OR osemail LIKE '" & search & "%')"
        End If

        Dim resUser: Set resUser = AdoSelect(query)

        If Not resUser.BOF Then
                resUser.MoveFirst
    ' Dynamische Erstellung von E-Mail-Objekten
                Do While Not resUser.EOF
                        emailAdress = resUser.Fields.Item("osemail").Value
                        If Len(emailAdress) > 5 Then
                            Set Email = CreateObject("Scripting.Dictionary")
                                Email.Add "name" , resUser.Fields.Item("name").Value
                                Email.Add "email" , resUser.Fields.Item("osemail").Value
                                Email.Add "source", source

                                ReDim Preserve Emails(UBound(Emails)+1)
                                Set Emails(UBound(Emails)) = Email
                        End If
                        resUser.MoveNext
                Loop
        End If

        End If

        Dim count: count = 0
        Dim Response: Set Response = XMLElement("emails")

    ' Abschliessende Generierung der XML-Antwort an das Dashlet
        With Response
                .Attr "size", UBound(Emails)+1
                For Each Email In Emails
                        With .Child("email")
                                .Attr "name", Email("name")
                                .Attr "email", Email("email")
                                .Attr "source", Email("source")
                        End With
                        count = count + 1
                Next
        End With

        Set GetEmails = Response.toDom
End Function
```

getFileName()

Hier wird der Namensvorschlag für ein enaio-Dokument generiert. Standardmässig gibt es die Pflicht- oder Systemfelder-Konkatenierung die einen Dateinamen generiert. getFileName hat dahingehend jedoch Vorrang.

Gibt diese Funktion nichts zurück, greift automatisch die vom System vorgegebenen Namenvorschläge. Es gilt folgende Hierarchie:

1. getFileName()
2. oder falls getFileName leer: Nach Pflichtfelder aus den Indexdaten
3. oder zumindest den Systemfeldern ObjectID und ObjectType die immer verfügbar sind.

```
' Funktion gibt vor, wie Anhänge benannt werden. Hier ist ein Text als Rückgabewert ausreichend, da immer nur 1 Doku
Function getFileName()
        fileName = "Alle Anhänge heissen jetzt gleich."

    getFileName = fileName
End Function
```

Nach oben

getObjectLocation()

Mit dieser Funktion kann der hierarchische Standort eines Dokuments festgestellt bzw. festgelegt werden. Wichtig ist die Funktion, wenn E-Mails beispielsweise in dem Standort als neues Dokument abgelegt werden sollen, aus dem heraus der Embedded Mailer verwendet wird (senden & ablegen-Funktion).

Sofern für die Ablage von E-Mails immer der selbe Schrank und das selbe Register verwendet wird, kann hier einfach ein String im korrekten Format (siehen unten im source-code) zurückgegeben werden.

Das Code-Beispiel zeigt, wie dynamisch festgestellt werden kann, in welchem Standort der Embedded Mailer gerade verwendet wird und legt z.B. diesen Standort als Ablageort für neue E-Mail-Dokumente fest.

```
' getObjectLocation nutzt den enaio DMS-Service, um den Standort eines Dokuments zu bestimmen
' Als Rückgabewert ist ein String ausreichend, der an den Embedded Mailer zurückgeliefert wird
Function getObjectLocation()
        Dim userName: userName = RC.InputParams.Value("$$$SwitchContextUserName$$$")
        Dim objectId: objectId = RC.InputParams.Value("objectId")
        Dim objectTypeId: objectTypeId = RC.InputParams.Value("objectTypeId")

        Dim Query: Set Query = XMLElement("DMSQuery")
        With Query
                .Attr "requesttype", "HOL"
                With .Child("Archive")
                        With .Child("ObjectType")
                                .Attr "id", objectTypeId
                                With .Child("Conditions")
                                        With .Child("ConditionObject")
                                                .Attr "id", objectTypeId
                                                With .Child("FieldCondition")
                                                        .Attr "internal_name", "OBJECT_ID"
                                                        .Attr "operator", "="
                                                        .Attr "system", "1"
                                                        .ChildWithContent "Value", objectId
                                                End With
                                        End With
                                End With
                                With .Child("Fields")
                                        .Attr "field_schema", "DEF"
                                        With .Child("Field")
                                                .Attr "internal_name", "SDSTA_ID"
                                                .Attr "system", "1"
                                        End With
                                        With .Child("Field")
```

```
                                End With
                                With .Child("Field")
                                        .Attr "internal_name", "SDREG_TYPE"
                                        .Attr "system", "1"
                                End With
                        End With
                End With
        End With
End With

Dim jobInputParameter: Set jobInputParameter = RC.NewJobsParams
Dim jobOutputParameter: Set jobOutputParameter = RC.NewJobsParams

jobInputParameter.Value("Flags") = 0
jobInputParameter.Value("Options") = 1
jobInputParameter.Value("Encoding") = "UTF-8"
jobInputParameter.Value("FieldSchema") = "MIN"
jobInputParameter.Value("XML") = Query.ToDom

RC.Jobs.dms.GetResultList jobInputParameter, jobOutputParameter

        Dim Result : Set Result = CreateObject("MSXML2.DOMDocument")
        Result.load(jobOutputParameter.Value("XML"))

        sdStaName = Result.selectSingleNode("//Archive").getAttribute("name")
        sdStaId = Result.selectSingleNode(".//Field[@internal_name=""SDSTA_ID""]").text
        sdRegId = Result.selectSingleNode(".//Field[@internal_name=""SDREG_ID""]").text
        sdRegType = Result.selectSingleNode(".//Field[@internal_name=""SDREG_TYPE""]").text

    ' Zusammengesetzter String als Rückgabewert mit Informationen über den Schrank und das Register in dem das ange
        getObjectLocation = sdStaId & "," & sdStaName & "," & sdRegId & "," & sdRegType
End Function
```

Nach oben

getPlaceholder()

Erstellt eine Liste nach eigenen Regeln aller verwendbaren Platzhalter für Nachrichten, Nachrichtenvorlangen und Layouts. Diese Regeln können beispielsweise system-, schrank- oder auch benutzerabhängig sein. Ebenso können Platzhalter statisch, nach einem enaio-Typ in einem Ordner oder auch generisch zum Beispiel beruhend auf dem Benutzerprofil ermittelt werden. Es stehen hier praktisch alle Möglichkeiten offen.

```
' getPlaceholder() erstellt die Liste der Platzhalter und liefert sie im XML-Format (vgl. getEmails()) zurück.
Function getPlaceholder()
    Dim userName: userName = RC.InputParams.Value("$$$SwitchContextUserName$$$")
    Dim placeholderList: placeholderList = Array()
    Dim Placeholder

' Statischer Platzhalter 1
Set Placeholder = CreateObject("Scripting.Dictionary")
Placeholder.Add "selector", "PLATZHALTERNAME1"
Placeholder.Add "replacement", "Platzhalterwert 1"
Placeholder.Add "description", "Platzhalter 1 Beschreibung"

ReDim Preserve placeholderList(UBound(placeholderList)+1)
Set placeholderList(UBound(placeholderList)) = Placeholder

' Statischer Platzhalter 2
Set Placeholder = CreateObject("Scripting.Dictionary")
Placeholder.Add "selector", "PLATZHALTERNAME2"
Placeholder.Add "replacement", "Platzhalterwert 2"
Placeholder.Add "description", "Platzhalter 2 Beschreibung"

ReDim Preserve placeholderList(UBound(placeholderList)+1)
Set placeholderList(UBound(placeholderList)) = Placeholder
```

```
        Dim Response: Set Response = XMLElement("placeholder")
        With Response
                .Attr "size", UBound(placeholderList)+1
                For Each Item In placeholderList
                        With .Child("replacement")
                                .Attr "selector", Item("selector")
                                .Attr "replacement", Item("replacement")
                                .Attr "description", Item("description")
                                .Content Placeholder(selector)
                        End With
                Next
        End With

        Set getPlaceholder = Response.toDom
End Function
```

Nach oben

persistEmail()

PersistEmail legt versendete E-Mails in einem Verzeichnis oder auch enaio®-Standort ab. Die Regeln für das Ablegen können frei definiert werden.

```
Function PersistEmail()
        ' Register:  Foldername to E-Mail-object typenames
        Dim folderMailTypes: Set folderMailTypes = CreateObject("Scripting.Dictionary")

        ' Beispiel: folderMailTypes.Add "Ordnername" , "E-Mail-Object-Typ: internal name"
        ' Festlegung welche enaio EMail Typen in welchem Ordner erlaubt sind. Nur hier können Emails abgelegt werder
        folderMailTypes.Add "Ordner1", "EMailTypSchrank1" 'internal_name
        folderMailTypes.Add "Ordner2", "EMailTypeSchrank2" 'internal_name

    Dim folderId: folderId = RC.InputParams.Value("folderId")
        Dim folderName: folderName = RC.InputParams.Value("folderName")

    ' Rückgabe als JSON-Objekt
        If folderMailTypes.Exists(folderName) Then
        Dim filePath: filePath = createEmailTempFile()

        If Len(filePath) > 0 Then
            Dim email: Set email = createEmail(filePath, folderMailTypes)

            RC.OutputParams.Value("objectId") = email.Value("ObjectID")
            RC.OutputParams.Value("objectTypeId") = email.Value("ObjectType")
            RC.OutputParams.Value("status") = true
            RC.OutputParams.Value("error") = ""
            RC.OutputParams.Value("message") = "Created enaio document for folder " & folderName & "(" & folderId &
        Else
            RC.OutputParams.Value("objectId") = 0
            RC.OutputParams.Value("objectTypeId") = 0
            RC.OutputParams.Value("status") = false
            RC.OutputParams.Value("error") = "EML-file in " & folderName & "(" & folderId & ") was not created."
            RC.OutputParams.Value("message") = ""
        End If
        Else
                RC.OutputParams.Value("objectId") = 0
                RC.OutputParams.Value("objectTypeId") = 0
                RC.OutputParams.Value("status") = false
                RC.OutputParams.Value("error") = "No folder " & folderName & "(" & folderId & ") related to e-mail c
                RC.OutputParams.Value("message") = ""
        End If
End Function

' Speichern des Base64 Kodierten EML Inhalts (E-Mail als Text wie sie auch via Outlook oder Thunderbird versendet wi
Function createEmailTempFile()
```

```
    Dim fso: Set fso = CreateObject("Scripting.FileSystemObject")
    Dim filePath: filePath = RC.ServerData.TempDir & "\" & fso.GetTempName & ".eml"

    Dim tempFile: Set TempFile = fso.CreateTextFile(FilePath, True, True)
    tempFile.write emlBase64 & vbCrLf
    tempFile.close()

    createEmailTempFile = filePath
End Function

' Erstellen eines enaio-E-Mail-Dokuments
Function createEmail(filePath, folderMailTypes)
    ' Ordner und Register festlegen, in dem ein E-Mail-Dokument in enaio erstellt werden soll

    ' Variante 1 "dynamisch": Systemfelder via HTTP-Request
    ' Schrank- und Registerinformationen kommen via HTTP-Anfrage und werden an Hand des zuletzt selektierten Dokume
    Dim folderId: folderId = RC.InputParams.Value("folderId")
        Dim folderName: folderName = RC.InputParams.Value("folderName")
        Dim registerId: registerId = RC.InputParams.Value("registerId")
        Dim registerTypeId: registerTypeId = RC.InputParams.Value("registerTypeId")

    ' Variante 2 "statisch": Systemfelder für festen Schrank & Register
    ' Schrank- und Registerinformationen werden fest im EmptyJob definiert. Alle E-Mail-Dokumente werden in diesem
'       Dim folderId: folderId = 2330
'       Dim folderName: folderName = "BeispielOrdner"
'       Dim registerId: registerId = 2420
'       Dim registerTypeId: registerTypeId = 6488064

    ' Variante 3 "kombiniert": Systemfelder für Schrank fest, Systemfelder für Register via HTTP-Request (zusätzlic
'       Dim folderId: folderId = 2330
'       Dim folderName: folderName = "BeispielOrdner"
'       Dim registerId: registerId = RC.InputParams.Value("registerId")
'       Dim registerTypeId: registerTypeId = RC.InputParams.Value("registerTypeId")


    Dim userName: userName = RC.InputParams.Value("$$$SwitchContextUserName$$$")

    ` Emailfelder via HTTP-Request
    Dim sender: sender = RC.InputParams.Value("from")
    Dim receiver: receiver = RC.InputParams.Value("to")
    Dim cc: cc = RC.InputParams.Value("cc")
    Dim bcc: bcc = RC.InputParams.Value("bcc")
    Dim sentOn: sentOn = RC.InputParams.Value("sentOn")
    Dim subject: subject = RC.InputParams.Value("subject")
    Dim message: message = RC.InputParams.Value("message")
    Dim messagePlainText: messagePlainText= RC.InputParams.Value("messagePlainText")
    Dim attachments: attachments = RC.InputParams.Value("attachments")

    ' Erstellen des enaio E-Mail-Dokuments via DMS Service
    Dim Query: Set Query = XMLElement("DMSData")

    With Query
        With .Child("Archive")
            With .Child("ObjectType")
                ' Vorsicht: In enaio 10.10. maintype nicht Abfragen. Führt zu einem falschen Ergebnis!
                .Attr "maintype", 6
                .Attr "internal_name", folderMailTypes.Item(folderName)
                With .Child("Object")
                    .Attr "folder_id", folderId
                    If registerId > 0 Then
                        .Attr "register_type", registerTypeId
                        .Attr "register_id", registerId
                    End If
                    With .Child("Fields")
                        With .Child("Field")
```

```
            End With
            With .Child("Field")
                .Attr "internal_name", "MAIL_TO"
                .Content receiver
            End With
            With .Child("Field")
                .Attr "internal_name", "MAIL_CC"
                .Content cc
            End With
            With .Child("Field")
                .Attr "internal_name", "MAIL_SUBJECT"
                .Content subject
            End With
            With .Child("Field")
                .Attr "internal_name", "MAIL_SUBMIT_TIME"
                .Content sentOn
            End With
        End With
        With .Child("Remarks")
            With .Child("RemarkText")
                .Attr "action", "INSERT"
                .Attr "color", "BLUE"
                .Content "E-Mail gesendet am von " & sender & " an " & receiver
            End With
        End With
    End With
End With
End With

    Dim jobInputParameter: Set jobInputParameter = RC.NewJobsParams
    Dim jobOutputParameter: Set jobOutputParameter = RC.NewJobsParams

    jobInputParameter.Value("Flags") = 0
    jobInputParameter.Value("File_0") = filePath
    jobInputParameter.Value("XML") = Query.ToDom

    RC.Jobs.dms.XMLInsert jobInputParameter, jobOutputParameter

    Set createEmail = jobOutputParameter
End Function
```

Nach oben

getTemplates()

Diese Funktion liefer eine Liste mit allen verfügbaren Nachrichtenvorlagen. Die Regeln sind frei definierbar. Es ist möglich statische oder Benutzerbezogene Nachrichtenvorlange auszulesen oder auch auf einen bestimmten Schrank zugeschnittene Nachrichtenvorlagen.

Die Regeln richten sich nach der Struktur, die hierfür eingesetzt werden soll.

Im Beispiel sind Nachrichtenvorlagen ein eigener enaio-Objekt-Typ dessen Indexdaten für die Nachrichtenvorlagen (Templates) verwendet werden. Dadurch könenn von jedem Benutzer beliebige Nachrichtenvorlagen in enaio definiert werden.

```
' Generiert eine XML-Liste von Nachrichtenvorlagen, die im Embedded Mailer verwendet werden können
Function getTemplates()
Dim search: search = RC.InputParams.Value("searchQuery")
Dim MaxResults: MaxResults = getMaxResults()
Dim department: department = getDepartment()

    ' Anfrage über DMS-Service nach Objekten vom Typ "MailingTemplate" (Schrankabhängig)
    Dim Query: Set Query = XMLElement("DMSQuery")
    With Query
        .Attr "requesttype", "HOL"
        With .Child("Archive")
```

```
                        With .Child("Conditions")
                                With .Child("ConditionObject")
                                        .Attr "operator", "AND"
                                        .Attr "internal_name", "MailingTemplate"
                                        With .Child("FieldCondition")
                                                .Attr "internal_name", "MailingTemplateSelection"
                                                .Attr "operator", "="
                                                .ChildWithContent "Value", search & "*"
                                        End With
                                        With .Child("FieldCondition")
                                                .Attr "internal_name", "MailingTemplateFirma"
                                                .Attr "operator", "="
                                                .ChildWithContent "Value", department
                                        End With
                                End With
                        End With
                        With .Child("Fields")
                                .Attr "field_schema", "DEF"
                                With .Child("Field")
                                        .Attr "internal_name", "MailingTemplateSelection"
                                End With
                                With .Child("Field")
                                        .Attr "internal_name", "MailingTemplateDescription"
                                End With
                                With .Child("Field")
                                        .Attr "internal_name", "MailingTemplateSubject"
                                End With
                                With .Child("Field")
                                        .Attr "internal_name", "MailingTemplateMessage"
                                End With
                        End With
                End With
        End With
End With

Dim jobInputParameter: Set jobInputParameter = RC.NewJobsParams
Dim jobOutputParameter: Set jobOutputParameter = RC.NewJobsParams

jobInputParameter.Value("Flags") = 0
jobInputParameter.Value("FileInfo") = 1
jobInputParameter.Value("Encoding") = "UTF-8"
jobInputParameter.Value("XML") = Query.ToDom

RC.Jobs.dms.GetResultList jobInputParameter, jobOutputParameter

Dim Result : Set Result = CreateObject("MSXML2.DOMDocument")
Result.load(jobOutputParameter.Value("XML"))

Dim Templates: Templates = Array()
Dim TemplateNodes, TemplateNode, Template
Dim count: count = 0

' Auslesen der Nachrichtenvorlagendaten
Set TemplateNodes = Result.SelectNodes("//ObjectType[@internal_name='MailingTemplate']/ObjectList/Object")
For Each TemplateNode In TemplateNodes
    Set Template = CreateObject("Scripting.Dictionary")
    Template.Add "selection" , TemplateNode.selectSingleNode(".//Field[@internal_name=""MailingTemplateSelectio
    Template.Add "description" , TemplateNode.selectSingleNode(".//Field[@internal_name=""MailingTemplateDescri
    Template.Add "subject" , TemplateNode.selectSingleNode(".//Field[@internal_name=""MailingTemplateSubject""]'
    Template.Add "message" , TemplateNode.selectSingleNode(".//Field[@internal_name=""MailingTemplateMessage""]'
    ReDim Preserve Templates(UBound(Templates)+1)
    Set Templates(UBound(Templates)) = Template

    count = count + 1
```

```
        End If
    Next

    ' Abschliessende Generierung der XML-Antwort an das Dashlet
    Dim Response: Set Response = XMLElement("templates")
    With Response
        .Attr "size", UBound(Templates)+1
        .Attr "department", department
        For Each Template In Templates
                With .Child("template")
                        .Attr "selection", Template("selection")
                        .Attr "description", Template("description")
                        .Attr "subject", Template("subject")
                        .Attr "message", Template("message")
                End With
        Next
    End With

    Set GetTemplates = Response.toDom

## End Function
```

Nach oben

Optionale Erweiterungen

Es gibt die Möglichkeit bestimmte Funktionalitäten bei Bedarf zu aktivieren, die sehr spezifischer Natur sind. Sie erlauben es, noch stärker auf die Bedürfnisse der Benutzer einzugehen und optimalere Anpassungen an die individuellen Anforderungen des Embedded Mailer zu ermöglichen.

Konfiguration und Freigabe

Die Konfiguration und somit auch die Freigabe erfolgt über die Konfigurationsdatei `Pfad\zu\Ihren\OptimalSystemsKomponenten\services\service-manager\config\embedded-mailer-prod.yaml` mittels des Abschnitts `context: extensions`.

```
mailing:
    ...
    extensions:
        # Angular/Typescript Frontend
        frontend:
            - name: [Eindeutiger Name der Erweiterung]
              # "Closure" ruft statische Funktion im Frontend oder Backend auf
              # "EmptyJob" stellt zusätzlich eine individualisierbare Anfrage an einen enaio&reg; EmptyJob
              type: [Closure | EmptyJob]
              paramters:
                - param1: [Wert]
                - param2: [Wert]
                ...
        # Java-Backend
        backend:
            - name: [Eindeutiger Name der Erweiterung]
              type: [Closure | EmptyJob]
              paramters:
                - param1: [Wert]
                - param2: [Wert]
                ...
    ...
```

Es können nur Erweiterungen verwendet werden, die auch implementiert sind. Es handelt sich hierbei nicht um generische Erweiterungen.

Erweiterung: allowSignature

Die Erweiterung `allowSignature` ermöglicht die individuell anpassbare Prüfung via enaio® EmptyJob, ob Signaturen generell oder unter bestimmten Umständen erlaubt sein sollen oder nicht.

```
mailing:
    ...
    extensions:
        frontend:
            - name: allowSignature
              type: EmptyJob
              expect: boolean
              paramters:
    ...
```

enaio® EmptyJob: Beispiel allowSignature

Im Beispiel wird die Option eine Signatur zu verwenden nur für einen bestimmten Benutzer erlaubt. Diese Funktionalität kann je enaio®-Installation individuell eingerichtet werden.

```
Function Main
        Dim Method: Method = RC.InputParams.Value("method")

        ' Exit Function If no Method was set
        If Method = "" Then
                RC.Logger.Error "Missing Method Parameter"
                Exit Function
        End If

        ' Execute Method
        Select Case Method
        Case "allowSignature" ' Closure
                RC.OutputParams.Value("result") = allowSignature()
        ...
        Case Else
                RC.Logger.Error "Unknown Method " & Method
                Exit Function
        End Select
End Function

' Rückgabeformat: boolean - siehe Konfiguration "expect"
Function allowSignature() ' Closure
        Dim userName: userName = RC.InputParams.Value("$$$SwitchContextUserName$$$")
    Dim allowed: allowed = False

    If userName == "administrator" Then
        allowed = True

        allowSignature = allowed
End Function
```

Nach oben