OPTIMAL SYSTEMS

enaio®

Softwaredokumentation
enaio® sharepoint-dms

Version 8.0

OPTIMAL SYSTEMS GmbH
Cicerostraße 26
D-10709 Berlin

03.12.2014
Version 8.0

Inhalt

Abbildungsverzeichnis

Tabellenverzeichnis

Zur Einführung

Das Handbuch liegt Ihnen als PDF-Datei vor. Die PDF-Datei wird in das Dokumentationsverzeichnis der enaio®-Installation bereitgestellt. Es kann mit einem PDF-Reader (z. B. dem Adobe Reader) am Bildschirm gelesen, ganz oder in Teilen ausgedruckt und schnell nach Begriffen durchsucht werden.

Das Handbuch beschreibt, wie die Schnittstelle enaio® sharepoint-dms installiert und konfiguriert wird. Außerdem erhalten Sie eine Übersicht über den Funktionsumfang der mitgelieferten Bausteine.

Über enaio® sharepoint-dms

enaio® sharepoint-dms stellt Funktionen für den Zugriff auf Daten und Dokumente aus enaio® in Microsoft SharePoint zur Verfügung. Damit die Lösungen an die Anforderungen des Kunden angepasst werden können, sind die SharePoint-Lösungen (WebParts) quelloffen und dokumentiert.

Auf Basis dieser Dokumentation und der Funktionen in SharePoint und enaio® ist die Realisierung von kundenindividuellen Lösungen mit SharePoint möglich.

Neben den lesenden Funktionen, die mit dem Produkt realisiert worden sind, können auch Dokumente von SharePoint nach enaio® übergeben werden. Dazu kann z. B. das zusätzliche Produkt enaio® sharepoint-archiv benutzt werden.

Systemarchitektur

Die folgende schematische Systemarchitektur stellt die notwendigen Komponenten für enaio® sharepoint-dms dar:

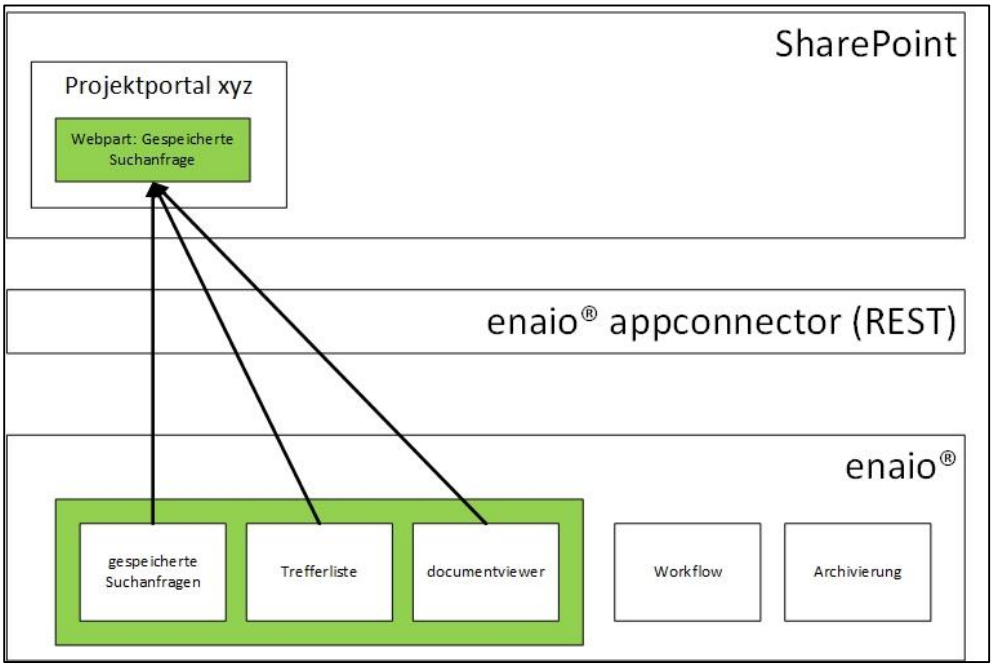


Abbildung 1 Systemarchitektur

Die Komponenten mit grünem Hintergrund sind Bestandteil der aktuellen Produktversion. Darüber hinaus können weiter Funktionen (z. B. Workflowintegration) durch den Kunden realisiert werden.

Für die Funktion Archivierung ist ein eigenes Produkt enaio® sharepoint-archiv entwickelt worden, womit Dokumente aus SharePoint regelbasiert in enaio® revisionssicher abgelegt werden können.

Voraussetzungen

Zur Installation des Produkts werden eine funktionsfähige SharePoint-Umgebung und ein funktionsfähiges enaio® vorausgesetzt. Zur Entwicklung der Lösung empfehlen wir die Einrichtung eines Entwicklungssystems. Im Detail benötigt das Produkt folgende Rahmenbedingungen:

Technische Rahmenbedingungen für SharePoint

- SharePoint-Editionen: ab Foundation
- SharePoint-Versionen: Die Lösung basiert auf SharePoint 2010 und ist damit sowohl auf SharePoint 2010 und 2013 Installationen verwendbar.
- Berechtigung für die Installation: SharePoint-Farm-Administrator
- Berechtigung für die Einrichtung der Lösung: SharePoint Site-Administrator
- Berechtigung für die Installation der Lösung: SharePoint Farm-Administrator

Die Lösung ist für die Installation in einer SharePoint „on premise“-Umgebung entwickelt worden.

Technische Rahmenbedingungen für das enaio®

Die Schnittstelle setzt folgende Rahmenbedingungen voraus:

- ein lauffähiges enaio®, Version 7.1 oder höher.
- Die Anzeige der Dokumente aus enaio® erfolgt in enaio® contentviewer. Daher muss dieser konfiguriert sein.
- enaio® appconnector (mit Patch, so dass XML-Darstellungen in enaio® appconnector als JSON-Daten abgerufen werden können). Weitere Informationen zu enaio® appconnector finden Sie im Handbuch dieser enaio®-Komponente.

Erfahrungen für Anwender und Integratoren

Für die Nutzung der Schnittstelle in SharePoint müssen die SharePoint-Anwender weder aktiv mit dem enaio® arbeiten noch Wissen aufweisen, wie enaio® funktioniert.

Die Administratoren und Integratoren benötigen zur Einrichtung der Funktionen und Code-Beispiele in SharePoint Erfahrungen in beiden Systemen. Vor allem zur Realisierung und Anpassung der mitgelieferten Beispiele ist tiefes Wissen in der SharePoint-Entwicklungsumgebung (z. B. Visual Studio, SharePoint Designer usw.) notwendig.

Lizenzmodul

Für die Aktivierung der Funktion wird das Lizenzmodul SPD (enaio® sharepoint-dms) benötigt. Lizenzen und Lizenzdateien verwalten Sie über den enaio® enterprise-manager.

Diese Lizenz ist über das Vertriebs-Team von OPTIMAL SYSTEMS oder über das OS-Service-Portal erhältlich.

Produktbestandteile

Die SharePoint-Schnittstelle ist ein beinhaltet verschiedene Komponenten, die als ZIP-Container ausgeliefert wird:

Komponente	Bezeichnung	Bemerkung
Webpart	enaio.sharepoint.dms.wsp	
Sourcecode	enaio.sharepoint.dms.source.V1.zip	Dieser ZIP-Container enthält das C#-Projekt mit allen Bestandteilen und einem Unterverzeichnis mit den Zusatzkomponenten.

Tabelle 1 Übersicht der Produktkomponenten

Installation

Notwendige Berechtigungen

Zur Einrichtung der Lösung ist die Berechtigung eines SharePoint-Farm-Administrators notwendig.

Auf der enaio®-Seite sind enaio®-Administrationsberechtigungen notwendig und der Zugriff auf die enaio®-Administrationskomponenten Voraussetzung.

Installationsablauf

Die Lösung wird als WSP-Datei (`enaio.sharepoint.dms.wsp`) bereitgestellt, die in SharePoint zu installieren ist. Dazu können folgende Schritte durchzuführt werden:

1. Anmelden am SharePoint Server als Farm-Administrator.
2. Kopieren der WSP-Datei in ein beliebiges Verzeichnis am SharePoint Server.
3. Starten der SharePoint Management Shell.
4. Innerhalb der SharePoint Management Shell in das Verzeichnis mit der Solution wechseln.
5. Solution zur Farm hinzufügen durch z. B. folgenden Befehl:
   ```
   stsadm.exe -o addsolution -filename enaio.sharepoint.dms.wsp
   ```
6. Solution einer Webanwendung bereitstellen durch z. B. folgenden Befehl:
   ```
   stsadm -o deploysolution -name enaio.sharepoint.dms.wsp -url
   {Url der Webanwendung} -immediate -allowGacDeployment -force
   ```
7. IISReset durchführen, z. B. durch folgenden Befehl:
   ```
   iisreset
   ```
8. Farm Feature enaio® sharepoint-dms Farm Feature in den Einstellungen der Farm Features der SharePoint Zentraladministration aktivieren.

Abbildung 2 Websitesammlungsfeature aktivieren

9. Site Feature enaio® sharepoint-dms in den Einstellungen der Webseitensammlung (Webseitensammlungsfeatures) aktivieren, in der das Webpart eingesetzt werden soll

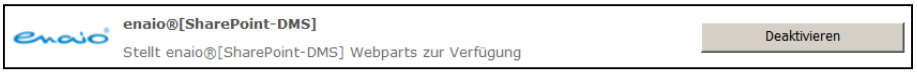


Abbildung 3 Webseitensammlungsfeature enaio® sharepoint-dms aktivieren

Konfiguration der Lösung

In der SharePoint Zentraladministration ist die URL zum enaio® appconnector auf Ebene einer Webanwendung bekannt zu machen. Hierzu sind folgende Schritte durchzuführen:

1. Öffnen der SharePoint Zentraladministration als Farm-Administrator.
2. Öffnen der Webseite zur Konfiguration der Webanwendungen.
3. Webanwendung markieren, für die das enaio® sharepoint-dms Feature genutzt werden soll.
4. Öffnen des enaio® sharepoint-dms-Konfigurationsdialogs über Klick auf die Schaltfläche Konfiguration enaio® sharepoint-dms (siehe Abbildung 4).

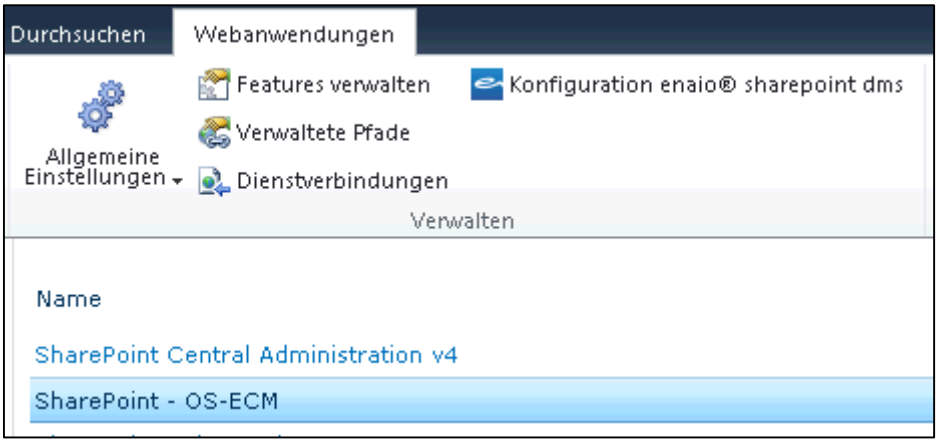


Abbildung 4 Öffnen des enaio® sharepoint-dms Konfigurationsdialogs

5. Damit öffnet sich ein Dialog, der für die Konfiguration der Verbindungsinformation zu enaio® notwendig ist:

Abbildung 5 Konfiguration verschiedener Parameter für den Zugriff auf enaio®

Mit dieser Aktion wird der SharePoint-Applicationpool dieser Anwendung auf allen Server der Farm recycelt. Sollte das nicht funktionieren, müssen die Pools per Hand recycelt werden.

Die Informationen zu den URLs für enaio® appconnector und enaio® contentviewer sind in enaio® enterprise-manager im Bereich **Servereigenschaften > Services** zu entnehmen:

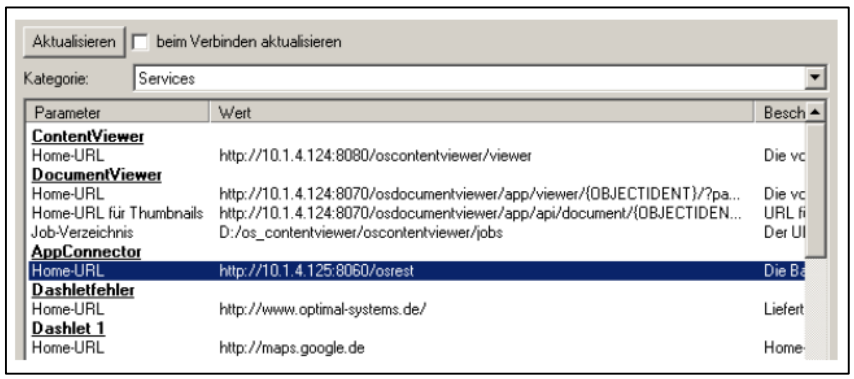


Abbildung 6 Übersicht der URLs

Parameter	Beschreibung	Beispiel1
enaio® appconnector URL	URL von enaio® appconnector	http://appcon.os.local:8060)
enaio® contentviewer 2	URL von enaio® contentviewer	http://appcon.os.local:8060/osdocumentviewer/app/viewer/{0}/?sessionGuid={1}
Gültigkeit des Anwendungscaches	Gültigkeit des SharePoint Anwendungscaches in Minuten3	

Tabelle 2 Konfiguration der Verbindung der Solution zu enaio®

6. Speichern der Einstellungen durch Betätigen der Schaltfläche **OK**[4].

[1] Die hier dargestellten Konfigurationsbeispiele müssen der Kundenumgebung angepasst werden.

2 Die Platzhalter „{0}“ und „{1}“ sind wie im Beispiel angegeben zu verwenden. Lediglich der Server- bzw. Domänenname der URL ist variabel verwendbar.

3 Dieser Wert ist aus Gründen der Performance etwas kleiner zu wählen, als die Cache-Dauer der Anmeldung am enaio® appconnector. Hierdurch werden sich wiederholende Anmeldevorgänge am enaio® appconnector vermieden.

4 Sollte eine Fehlermeldung beim Speichern der Einstellungen erscheinen, hat das Recycling des bzw. der Application Pools in der SharePoint Farm fehlgeschlagen. In diesem Fall ist der Application Pool der Anwendung auf dem oder den Web Frontend Servern per Hand zu recyceln.

Installation des Webparts

Das Webpart enaio® sharepoint-dms Gespeicherte Suchabfragen (siehe Abbildung 8) ist zunächst von einer Person mit Administrationsrechten auf einer SharePoint-Seite einzubinden.

Konfiguration des Webparts

Danach können die Webpart Eigenschaften des Abschnitts „enaio® sharepoint-dms Settings“ gemäß nachstehender Tabelle über den Konfigurationsdialog eingerichtet werden:

Webpart-Parameter	Beschreibung
enaio® Nutzer	Name des enaio®-Nutzers, unter dessen Anmeldung die Suchabfrage in enaio® ausgeführt werden soll.
enaio® Password	Passwort des enaio®-Nutzers, unter dessen Anmeldung die Suchabfrage in enaio® ausgeführt werden soll.[5]
Filter Prefix	Suchabfragen sind in enaio® mit einem Namen versehen, der in der Trefferliste des Webparts erscheint (siehe Abbildung 8). Wird in diesem Feld ein Präfix angegeben, werden nur die Suchabfragen aus enaio® übernommen (gefiltert), die in ihrem Namen mit dem angegebenen Präfix beginnen. Ist kein Präfix angegeben, werden alle Treffer von Suchabfragen aus enaio® übernommen.

Tabelle 3 Konfiguration des WebParts

Abbildung 7 Konfigurationsdialog des Webparts

[5] Die aktuelle Version der Lösung meldet den Nutzer programmatisch über die HTTP BASIC Authentifizierung (RFC2617) am REST Webservice vom enaio® appconnector an. Ist am appconnector der freie Zugang für den Nutzer konfiguriert, ist das Passwort an dieser Stelle leer zu lassen.

Anwendungsszenarien

Zur Demonstration der Integrationsfähigkeit und Anbindung von enaio® in SharePoint sind die folgenden Funktionen realisiert worden:

1. Anzeige bestimmter gespeicherten Suche aus enaio®
2. Anzeige der Trefferliste zu einer gespeicherten Suche in einer SharePoint-Liste
3. Anzeige von Dokumenten aus einer Trefferliste im formatunabhängigen enaio® contentviewer

Szenario 1: Darstellung der gespeicherten Anfragen (Webpart)

Zur Auswahl einer gespeicherten Anfragen enthält die Lösung ein Webpart enaio® sharepoint-dms Gespeicherte Suchabfragen, das , wenn sowohl die Lösung als auch das Webpart konfiguriert ist, folgendes Erscheinungsbild besitzt, sobald es in eine SharePoint Seite integriert ist:

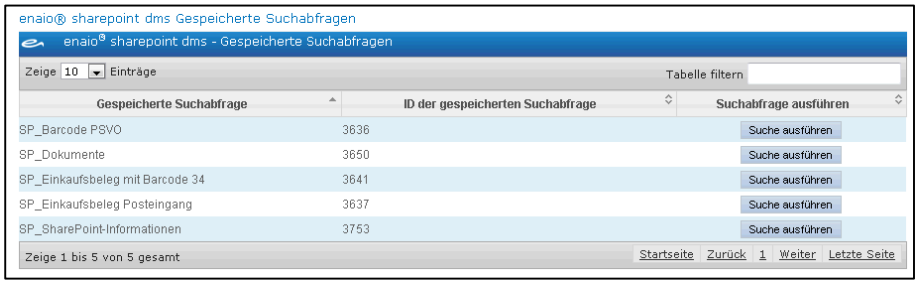


Abbildung 8 Webpart enaio® sharepoint-dms Gespeicherte Suchabfragen

Voraussetzung für diese Ansicht ist, dass der konfigurierte enaio®-Nutzer auch gespeicherte Anfragen hat (siehe auch Tabelle 3).

Zur Herstellung der Funktionsfähigkeit des Webparts ist die Lösung zunächst in SharePoint zu installieren (siehe Kapitel „Installation“), zu konfigurieren (Abschnitt siehe Kapitel „Konfiguration der Lösung“) und letztendlich die enaio® Zugangsdaten am Webpart einzustellen (siehe Kapitel „Konfiguration des Webparts“).

Beschreibung des Aufbaus und des Inhalts des Webparts

Die erste Spalte „gespeicherte Suchabfrage“ beinhaltet alle gespeicherten Abfragen, die dem Filterkriterium und den Benutzerdaten entsprechen. Die nächste Spalte stellt die ID der gespeicherten Suche aus enaio® dar. In der dritten Spalte wird eine Schaltfläche dargestellt, welche die gespeicherte Suche in der Zeile der Schaltfläche ausführt.

Bei einer großen Liste kann die Anzahl der Einträge über die Eingabe eines Begriffs im Feld „Tabelle filtern“ des Webparts bestimmte eingeschränkt werden. Außerdem kann die Anzahl der Einträge über die Einstellung „Zeile xx Einträge“ in der gleichen Zeile, in der auch die Filterung eingestellt werden kann, beeinflusst werden.

Im Fuß des Webparts befinden sich noch Navigationsfunktionen innerhalb der Trefferliste.

Szenario 2: Anzeige der Trefferliste

Nach der Ausführung der gespeicherten Anfrage wird eine Trefferliste in SharePoint erzeugt. Diese stellt alle zum Zeitpunkt der Ausführung existierende Treffer dar und ermittelt dynamisch alle Indexdaten aus enaio® und zeigt diese in Spalten an.

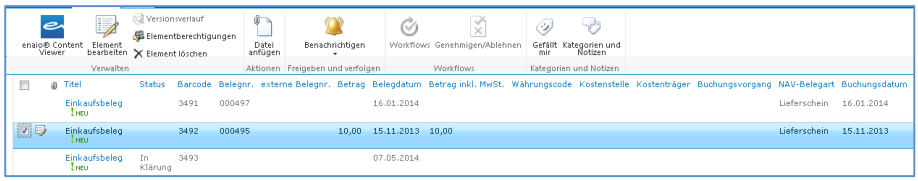


Abbildung 9 Trefferliste zu einer Suchanfrage

Der Nutzer der Website kann nun alle Konfigurationsmöglichkeiten von SharePoint auf diese Liste anwenden, insbesondere Sortierung und Filterung der Liste. Die Liste bleibt auf der Website so lange existent, bis eine weitere bzw. neue Suchabfrage ausgeführt wird.

Szenario 3: Aufruf und Anzeige eines Dokuments

Die Anzeige eines Dokuments erfolgt durch den Aufruf von enaio® contentviewer.

Dazu markiert der Anwender einen Listeneintrag (siehe Punkt 1). Damit erweitert sich die Ribbonleiste um die Schaltfläche enaio® contentviewer (siehe Punkt 2).

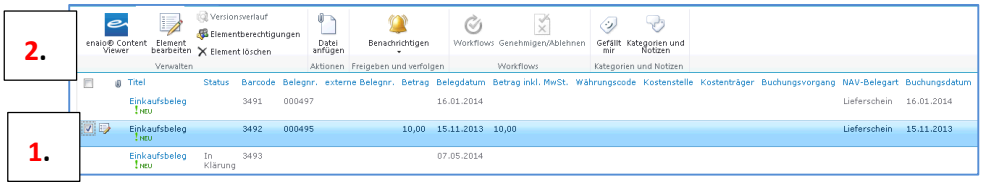


Abbildung 10 Darstellung der Trefferliste

Durch einen Klick auf die Schaltfläche wird enaio® contentviewer geöffnet (siehe Abbildung 11). Damit öffnet sich ein neues SharePoint Fenster, in welchem das Dokument aus enaio® angezeigt wird.

> Es kann immer nur ein Dokument angezeigt werden.

Abbildung 11 Darstellung des Dokuments mit enaio® documentviewer

Innerhalb des Dokuments kann die Volltextsuche vom enaio® documentviewer genutzt werden. Außerdem kann über die Auswahl „Indexdaten“[6] alle Indexdaten, die Historie und weitere Informationen zum Dokument angezeigt werden.

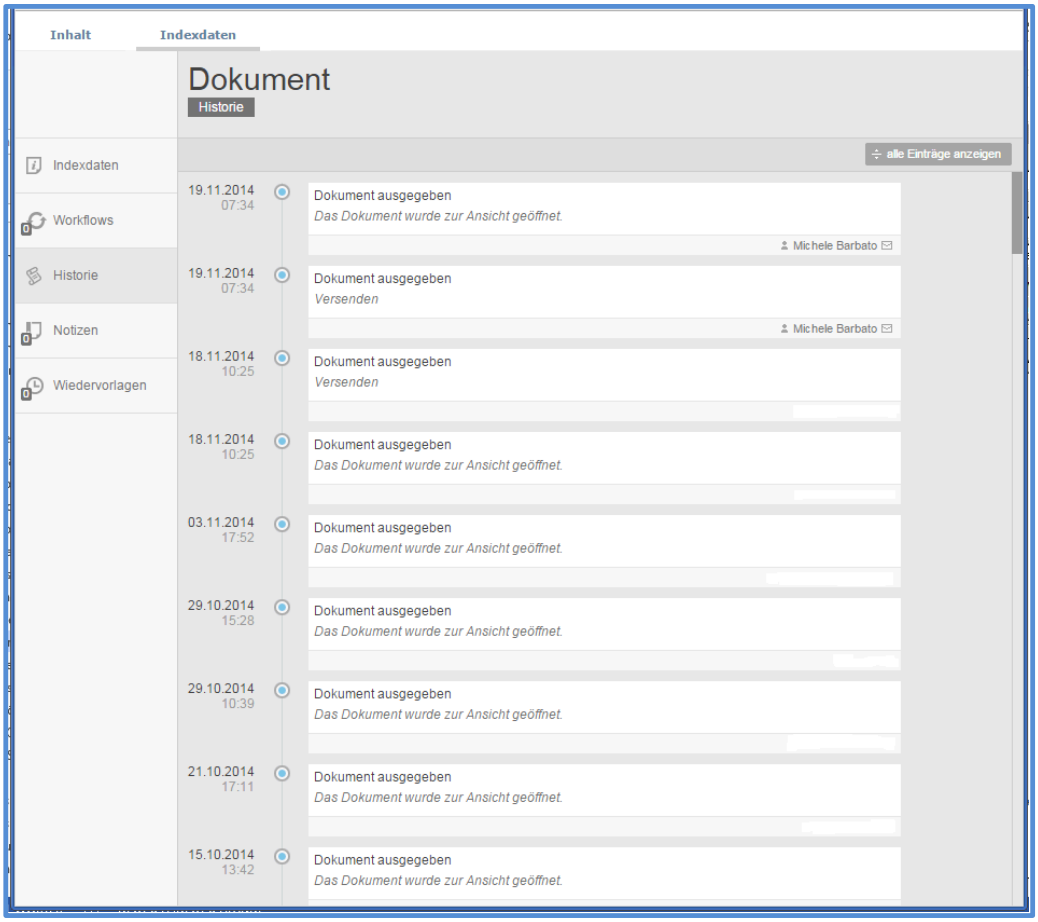


Abbildung 12 Darstellung der Informationen mit enaio® detailsviewer - Historie[6]

[6] Diese Ansicht des enaio® detailsviewer stammt aus der Version enaio® 8. In anderen Versionen kann dieser Dialog anders aussehen oder gar nicht verfügbar sein.

Dokumentation der Lösung

Die Lösung basiert auf der Erstellung einer SharePoint 2010-Projekts im MS Visual Studio 2010.

Struktur der Lösung

Die Lösung hat, wenn sie im Visual Studio geöffnet wird, die folgende Struktur:

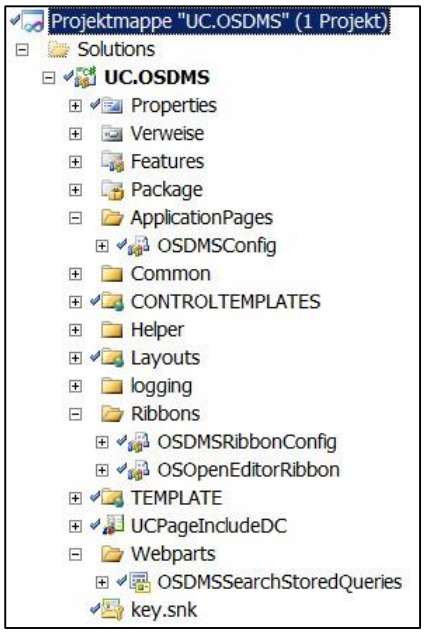


Abbildung 13 Struktur der Lösung in Visual Studio

Elemente der Lösung

Die Elemente der Lösung werden in der folgenden Tabelle zusammengefasst:

Element	Beschreibung
Verweise	Verweise auf die Drittbibliotheken, die von der Lösung verwendet werden. Einzige Bibliothek, die über den Standard hinausgeht, ist die frei erhältliche Bibliothek Newtonsoft.Json[7], die eine Deserialisierung der von enaio® appconnector empfangenen Json-Daten in C# Objekte vornimmt.
Features	Die Features der Lösung: **enaio® sharepoint-dms Farm Feature** Feature zur Provisionierung des enaio® Loggers im SharePoint Diagnostic Logging und Bereitstellen des enaio® [SharePoint-DMS]-Konfigurationsdialogs in der SharePoint Zentraladministration. **enio® sharepoint-dms Site Feature** Feature mit dem Webpart und den Anpassungen für die SharePoint-Ribbons
Application-Pages/OSDMSConfig	ASPX-Seite für die enaio®-Konfiguration in der Zentraladministration (Abbildung 3).
Common	Ordner mit Objektklassen zur Deserialisierung der enaio® JSON Daten.
ControlTemplates	SharePoint Mapped Folder mit zwei UserControls: **UCOSUIRibbonDelegate.ascx** Delegate Control als Eventhandler des enaio® Buttons im SharePoint Ribbon für List items
	UCPageInclude.ascx Delegate Control zur Einbettung der benötigten Webressourcen in die SharePoint Seite (CSS Dateien, Javascripte)
Helper	Hilfsklassen
Layouts	SharePoint Mapped Folder mit den Webressourcen der Lösung (CSS Dateien, JavaScripte, Bilder, …)

[7] siehe JSON.NET (http://json.codeplex.com/)

Element	Beschreibung
Logging	Ordner mit der Klasse für ein eigenes enaio®-SharePoint Diagnostic Logging
Ribbons	Ordner mit den Definitionen der enaio® Erweiterungen der SharePoint Ribbons
Template	SharePoint Mapped Folder zur Aufnahme des enaio® Icons
UCPageIncludeDC	Element mit der Definition des Delegate Controls **UCPageInclude** als Einbettung in den SharePoint **Platzhalter Additional-PageHead**
Webparts	Ordner mit dem Webpart **enaio® sharepoint-dms Gespeicherte Suchabfragen**. Der CodeBehind zum Webpart (`EnaioSearchStoredQueriesUserControl.ascx.cs`) enthält die Businesslogik zur Kommunikation mit dem enaio® Rest Service (enaio® appconnector).

Tabelle 4 Elemente der Lösung enaio® sharepoint-dms

Hinweise zum Quellcode

Die Konvertierung von Json-Daten in C# Objekte (Serialisierung/Deserialisierung) kann über mehrere Arten erfolgen. Diese Lösung bedient sich dem Opensource Projekt JSON.NET. Die Deserialisierung ist in der Methode FetchDataFromOSRest() gekapselt.

Ausgangspunkt der Verarbeitungslogik im Webpart ist das Page Load Ereignis der ASPX Seite:

```
31          /// <summary>
32          /// Page Load Ereignis der Webseite
33          /// </summary>
34          /// <param name="sender"></param>
35          /// <param name="e"></param>
36          protected void Page_Load(object sender, EventArgs e)
37          {
38
39              if (CheckForLogin())
40              {
41                  /*
42                   * Liste der Stored Queries holen
43                   */
44                  FetchStoredQueries();
45
46                  if (HttpContext.Current.Request.Params["ossqid"] != null)
47                  {
48                      /*
49                       * Dokumente der gespeicherten Suchabfrage holen
50                       */
51                      if (FetchStoredDocuments(HttpContext.Current.Request.Params["ossqid"]))
52                      {
53                          //Auf die Seite der Liste weiterleiten
54                          RedirectToDocList();
55                      }
56                      else
57                      {
58                          litOSResult.Text =
59                              "Problem beim Holen von Dokumenten aus OS|ECM. Siehe SharePoint Diagnostic Logging!";
60                          litOSResult.Visible = true;
61                      }
62                  }
63                  else
64                  {
65                      litOSResult.Visible = false;
66                  }
67              }
68          }
```

Abbildung 14 Verarbeitungslogik „Page Load Ereignis“ des enaio® Webparts

Zunächst wird das Login des am Webpart konfigurierten Nutzers geprüft (`CheckForLogin()`) bzw. durchgeführt. Anschließend werden die gespeicherten Suchabfragen vom enaio® Rest Service abgerufen und als HTML Tabelle gerendert (`FetchStoredQueries()`).

Wird eine konkrete Suchabfrage aufgerufen (dadurch, dass der Nutzer im Portal eine Suchabfrage des Webparts anklickt), werden alle Dokumente dieser Suchabfrage am enaio® Rest Service angefragt und in eine SharePoint Liste geschrieben (`FetchStoredDocuments()`). Sind keine Fehler aufgetreten, leitet der Code den Request an die Seite mit der soeben befüllten SharePoint Liste weiter (`RedirectToDocList()`).

Bei aufgetretenen Fehlern wird das Control zur Anzeige der Tabelle ausgeblendet und stattdessen ein Fehlertext angezeigt.

Wesentlicher Teil zur Kommunikation des Webparts mit enaio® appconnector sind drei Aufrufe:

- Aufruf aller Stored Queries
- Aufruf einer Stored Query
- Aufruf eines Dokuments

Aufruf aller Stored Queries

Der Aufruf aller gespeicherten Suchabfragen erfolgt in der Methode `FetchStoredQueries()`.

Die URL zum Aufruf des REST-Service wird so zusammengebaut, dass sie im enaio® appconnector die Ressource `/osrest/api/documents/storedqueries` aufruft (siehe auch Abbildung 15, Zeile 120).

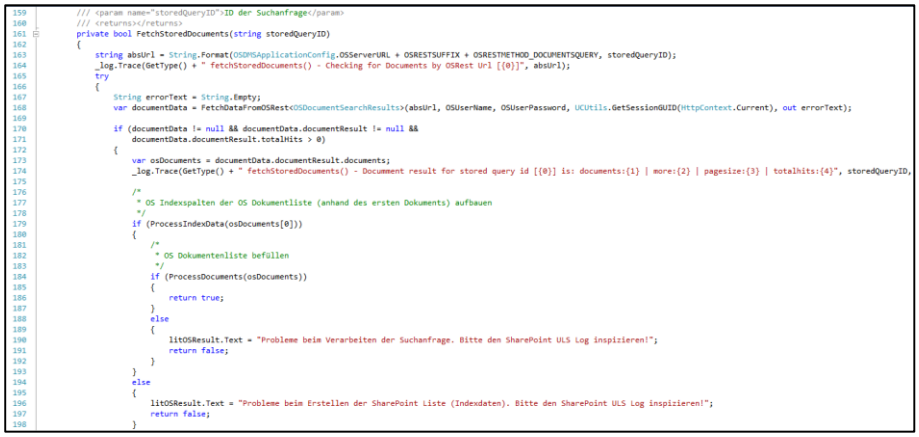
```
112         /// <summary>
113         /// Holt alle gespeicherten Suchabfragen vom OSRest Service und baut eine HTML Ergebnistabele zusammen,
114         /// die in die Seite des Webparts gerendert wird.
115         /// der Aufruf an den OS Rest Service erfolgt über den URL [/osrest/api/documents/storedqueries].
116         /// </summary>
117         /// <returns></returns>
118         private bool FetchStoredQueries()
119         {
120             string absUrl = OSDMSApplicationConfig.OSServerURL + OSRESTSUFFIX + OSRESTMETHOD_STOREDQUERIES;
121             _log.Trace("fetchStoredQueries() - Checking for stored Queries in OSRest URL [{0}]", absUrl);
122             string errorText = String.Empty;
123             var sessionData = FetchDataFromOSRest<OSStoredQueries>(absUrl, OSUserName, OSUserPassword, UCUtils.GetSessionGUID(HttpContext.Current), out errorText);
124 
125             if (sessionData != null && sessionData.StoredQueries != null)
126             {
127                 StringBuilder buf = new StringBuilder();
128                 buf.Append("<table id='ucstoredqueries' border='0' cellpadding='0' cellspacing='0'>");
129                 buf.Append("<thead><tr><th>Gespeicherte Suchabfrage</th><th>ID der gespeicherten Suchabfrage</th><th>Suchabfrage ausführen</th></tr></thead>");
130                 var storedQueries = sessionData.StoredQueries;
131                 buf.Append("<tbody>");
132                 foreach (var query in storedQueries)
133                 {
134                     String queryName = UCUtils.GetUTF8String(query.name);
135                     if (String.IsNullOrEmpty(OSFilterPrefix) || (!String.IsNullOrEmpty(OSFilterPrefix) && queryName.StartsWith(OSFilterPrefix)))
136                     {
137                         buf.Append("<tr><th>" + queryName + "</th><th>" + query.id +
138                                    "</th><th><input type='button' class='quicksearchbutton' value='OS|ECM Suche ausführen' rel='" +
139                                    UCUtils.ReplaceOrAddHttpParam(Page.Request.Url.ToString(), "ossqid", query.id.ToString()) +
140                                    "' /></th></tr>");
141                     }
142                 }
143                 buf.Append("</tbody>");
144                 buf.Append("<tfoot></tfoot></table>");
145 
146                 litOSContent.Text = buf.ToString();
147                 return true;
148             }
149             litOSContent.Text = "Keine gespeicherten Suchanfragen gefunden";
150             return false;
151         }
```

Abbildung 15 Methode zum Abruf der Suchabfragen und Bau einer HTML-Tabelle

Aufruf einer Stored Query

Der Aufruf einer gespeicherten Suchabfrage erfolgt in der Methode `FetchStoredDocuments()`.

Die URL zum Aufruf des Rest Services wird so zusammengebaut, dass sie am AppConnector die Ressource `/osrest/api/documents/storedqueries/{ID der Suchabfrage}` aufruft (siehe Abbildung 16, Zeile 161).

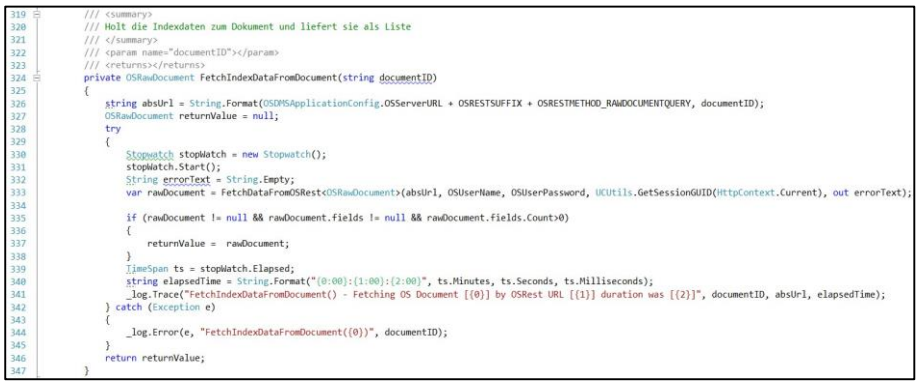


```
159         /// <param name="storedQueryID">ID der Suchanfrage</param>
160         /// <returns></returns>
161         private bool FetchStoredDocuments(string storedQueryID)
162         {
163             string absUrl = String.Format(OSDMSApplicationConfig.OSServerURL + OSRESTSUFFIX + OSRESTMETHOD_DOCUMENTSQUERY, storedQueryID);
164             _log.Trace(GetType() + " fetchStoredDocuments() - Checking for Documents by OSRest Url [{0}]", absUrl);
165             try
166             {
167                 String errorText = String.Empty;
168                 var documentData = FetchDataFromOSRest<OSDocumentSearchResults>(absUrl, OSUserName, OSUserPassword, UCUtils.GetSessionGUID(HttpContext.Current), out errorText);
169 
170                 if (documentData != null && documentData.documentResult != null &&
171                     documentData.documentResult.totalHits > 0)
172                 {
173                     var osDocuments = documentData.documentResult.documents;
174                     _log.Trace(GetType() + " fetchStoredDocuments() - Document result for stored query id [{0}] is: documents:{1} | more:{2} | pagesize:{3} | totalhits:{4}", storedQueryID,
175 
176                     /*
177                      * OS Indexspalten der OS Dokumentliste (anhand des ersten Dokuments) aufbauen
178                      */
179                     if (ProcessIndexData(osDocuments[0]))
180                     {
181                         /*
182                          * OS Dokumentenliste befüllen
183                          */
184                         if (ProcessDocuments(osDocuments))
185                         {
186                             return true;
187                         }
188                         else
189                         {
190                             litOSResult.Text = "Probleme beim Verarbeiten der Suchanfrage. Bitte den SharePoint ULS Log inspizieren!";
191                             return false;
192                         }
193                     }
194                     else
195                     {
196                         litOSResult.Text = "Probleme beim Erstellen der SharePoint Liste (Indexdaten). Bitte den SharePoint ULS Log inspizieren!";
197                         return false;
198                     }
```

Abbildung 16 Methode zum Abruf einer Suchabfrage (und Bau der Liste)

Aufruf eines Dokuments

Der Aufruf eines Dokuments erfolgt in der Methode `FetchIndexDataFromDocument()`. Die URL zum Aufruf des Rest Services wird so zusammengebaut, dass sie am AppConnector die Ressource `/osrest/api/documents/raw/{docId}?format=json` aufruft (siehe Abbildung 17, Zeile 326).



```
319         /// <summary>
320         /// Holt die Indexdaten zum Dokument und liefert sie als Liste
321         /// </summary>
322         /// <param name="documentID"></param>
323         /// <returns></returns>
324         private OSRawDocument FetchIndexDataFromDocument(string documentID)
325         {
326             string absUrl = String.Format(OSDMSApplicationConfig.OSServerURL + OSRESTSUFFIX + OSRESTMETHOD_RAWDOCUMENTQUERY, documentID);
327             OSRawDocument returnValue = null;
328             try
329             {
330                 Stopwatch stopWatch = new Stopwatch();
331                 stopWatch.Start();
332                 String errorText = String.Empty;
333                 var rawDocument = FetchDataFromOSRest<OSRawDocument>(absUrl, OSUserName, OSUserPassword, UCUtils.GetSessionGUID(HttpContext.Current), out errorText);
334 
335                 if (rawDocument != null && rawDocument.fields != null && rawDocument.fields.Count>0)
336                 {
337                     returnValue =  rawDocument;
338                 }
339                 TimeSpan ts = stopWatch.Elapsed;
340                 string elapsedTime = String.Format("{0:00}:{1:00}:{2:00}", ts.Minutes, ts.Seconds, ts.Milliseconds);
341                 _log.Trace("FetchIndexDataFromDocument() - Fetching OS Document [{0}] by OSRest URL [{1}] duration was [{2}]", documentID, absUrl, elapsedTime);
342             } catch (Exception e)
343             {
344                 _log.Error(e, "FetchIndexDataFromDocument({0})", documentID);
345             }
346             return returnValue;
347         }
```

Abbildung 17 Methode zum Abruf der Rohdaten eines Dokuments

Anpassung des UI für die Trefferliste der Suchanfragen

Die HTML-Tabelle der Suchanfragen wird clientseitig durch das frei erhältliche jQuery PlugIn „DataTables“ gerendert, so dass Elemente wie UI Styling, Paging, Sortierung usw. nicht „per Hand“ programmiert werden müssen. Das Plugin unterstützt das „ThemeRolling“, so dass über die Webseiten des jQuery UI Plugins ein eigenes Theme erstellt und in die Lösung übernommen werden kann. Das neue Theme kann im SharePoint Mapped Folder `Layouts/UC.OSDMS/css/jui_themes` eingefügt und im `UserControl UCPageInclude.asxc` angepasst werden (siehe Abbildung 18, Zeile 10).

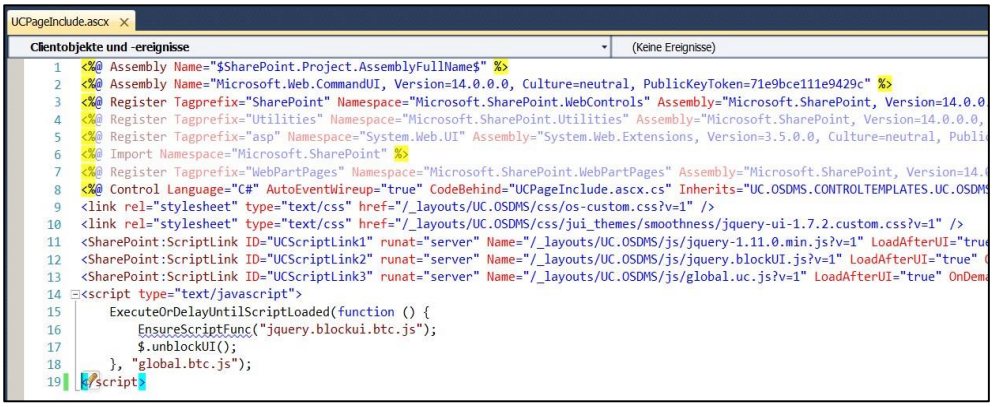


Abbildung 18 Anpassen des Themes über das UserControl UCPageInclude.ascx

Troubleshooting und Fehleranalyse

Die Lösung verfügt über ein ausgeprägtes Logging. Die Intensität des Loggings kann in der SharePoint Zentraladministration (Monitoring / Reporting / Diagnostic Logging) am Produkt enaio® sharepoint-dms eingestellt werden (siehe Abbildung 19):

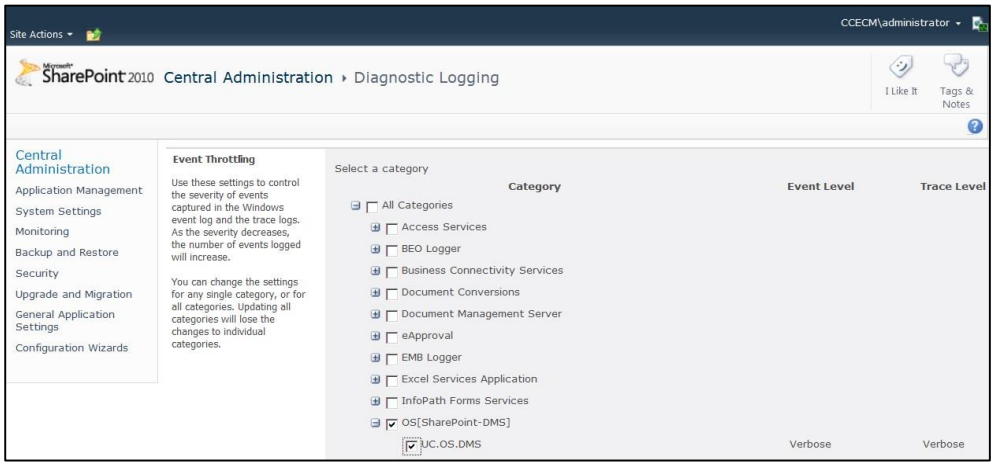


Abbildung 19 Einstellen des Log-Levels am SharePoint Diagnostic Logging

Die in der Lösung angewendeten Stufen sind Verbose (Trace), High (Fehler) und Monitorable (Info).

Bei Fehlern ist der SharePoint ULS Log zu konsultieren. Es wird empfohlen, hierfür den frei erhältlichen ULSViewer zu nutzen und in der Spalte Product auf enaio[SharePoint-DMS] zu filtern (siehe Abbildung 20).

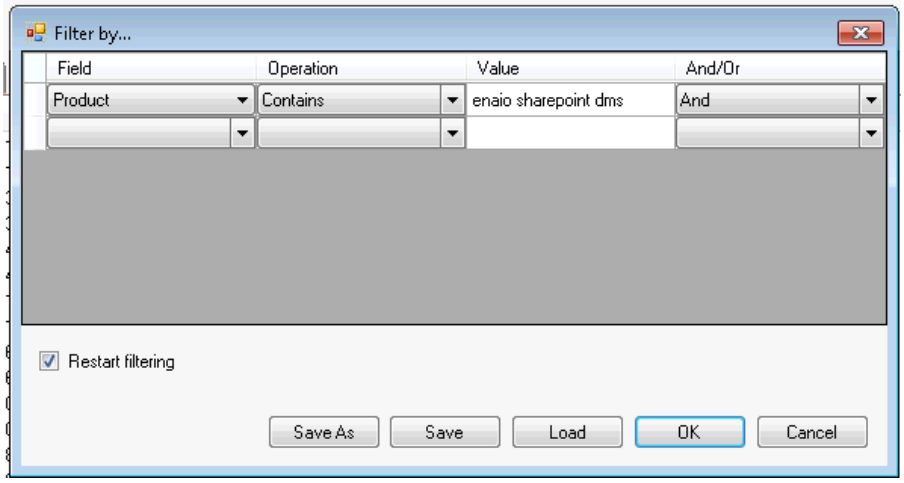


Abbildung 20 Nutzung des ULS Viewers und Filtern auf „enaio sharepoint-dms“

enaio®-Dokumentationen

Die in der folgenden Tabelle aufgelisteten Dokumentationen finden Sie im Dokumentationsverzeichnis …`clients\admin\Dokumentation`.

Dokumentation	Inhalt
enaio® appconnector	Beschreibung der Funktion von enaio® appconnector: enaio® appconnector ist ein Kerndienst von enaio® und stellt eine REST-Schnittstelle (Representational State Transfer) bereit. Damit ist ein flexibler HTTP-Zugriff auf Index- und Dokumentendaten in enaio® möglich.

Tabelle 5 Übersicht der enaio®-Dokumentationen