ALLAN THRAEN | 1 years ago | PDF |

Vision Demos & Prototypes Tips and Tricks Optimizely (Episerver) Addon Development

CONTENT REPORT GENERATOR V.2.

When you are about to migrate a site from CMS 11 to CMS 12, it's not unusual to want to do some rewriting and re-implementation to clean up the architecture as part of the project. When you do that (and in many other cases) it can be incredibly helpful to have a handy overview of your content, it's usage and your visitor groups and their usage. Here is an easy way to get that.

A long time ago, I made a simple Scheduled Job, called the Content Report Generator and shared the code for it here on this blog. It was quick and dirty code with a terrible architecture - and not that easy to expand. However, it solved a real problem: provide a way to get an overview of all your content in a way where you can work with it.

It did this by simply creating an excel file with a list of your content + some very valuable collected meta-information - like where it was used and who was supposed to approve it. Usually I'm no fan of excel - but it's a format that everybody usually can read - and it's even pretty easy to do filtering, grouping, sorting and similar actions - so it's pretty perfect for this purpose.

Recently, I've been working on a couple of really complex site-rewrites onto CMS 12 - and it has come in very useful to be able to extract similar reports - however I've started to notice an extensive use of Visitor Groups in many sites - as well as a common architecture where pages have a main content area and the actual content is in blocks - or even in blocks in blocks. This makes it increasingly difficult to spot which blocks and visitors groups are used on which urls.

So - I've rewritten and extended the old scheduled job. First of all, to get the code up to a somewhat better quality, making it more modularized so it's easy to add more data sheets - but also to support data about visitor groups, criteria and their use on the site.

To achieve this I used similar code to that of the helper methods I recently shared, Optimizely CMS: List content recursively on a page - and list the visitor groups used, and a bit of other magic.

The job still creates a content report and puts it in a folder in your sites assets ("Internal Reports") ready for download.

Here is an example of an output report from my test-alloy site (so not terribly interesting) - however I'm sure you can imagine what you might get out of this on your own site: Report 2022-05-20 13_57.xlsx.

It currently contains 3 sheets:

1. Content Details. The list of content items (1 per ID+language) and essential meta-data. But also a list of where it's used, how many versions exist, who it's waiting on for review, and a list of which visitor groups it's using.
2. Visitor Group Details. A list of each visitor group's criteria - and all the details regarding both the visitor group and the usage of that criteria. Could be used to spot duplicates or find out which groups are using which criteria.
3. Visitor Group Usage. A list of where the visitor groups are used on the site. If a visitor group is used in a block, then it will both be listed for the block, but also for all the pages that block is used on.

The code for doing this in CMS 11 is below (and who knows - I might eventually make a package out of it) - but it shouldn't be hard to customize it and use it in CMS 12 as well.

Future

I obviously quickly realized that it's even more fun to have this in PowerBI than 'just' Excel - and even though this format already supports some pretty nice PowerBI views, I think it could be even more fun to add more sheets and normalized data, where relationships can be set up in PowerBI. So hopefully I'll soon find time to do that as well - and maybe there'll be yet another blog post in this series...

```
using EPiServer;
using EPiServer.Approvals;
using EPiServer.Approvals.ContentApprovals;
using EPiServer.Core;
using EPiServer.DataAbstraction;
using EPiServer.Personalization.VisitorGroups;
using EPiServer.Web;
using EPiServer.Web.Mvc.Html;
using Newtonsoft.Json;
using System;
using System.Collections.Generic;
using System.Globalization;
using System.Linq;
using System.Web.Mvc;

namespace CodeArt.Episerver.ContentReport
{

    public class ContentAnalyzer
    {
        private readonly IContentRepository _repo;
        private readonly IContentVersionRepository _vrepo;
        private readonly ISiteDefinitionRepository _sites;
        private readonly IContentTypeRepository _types;
        private readonly ISiteDefinitionResolver _resolver;
        private readonly IApprovalDefinitionRepository _apdefrepo;
        private readonly IVisitorGroupRepository _vgrepo;
        private readonly IContentSoftLinkRepository _slrepo;
```

```
    private readonly UrlHelper _url;
    private readonly int _trashid;


    public List<ContentDetails> ContentDetails{ get; set; }

    public List<VisitorGroupUse> VisitorGroupUses { get; set; }

    public List<VisitorGroupDetails> VisitorGroupDetails { get; set; }

    protected Dictionary<ContentReference, List<ContentReference>> usedBy { get; set; }



    public ContentAnalyzer(IContentRepository repo, IContentVersionRepository vrepo, ISiteDefinitionRepository sites, IContentTypeRepository types, ISiteDef
    {
        _repo = repo;
        _vrepo = vrepo;
        _sites = sites;
        _types = types;
        _resolver = resolver;
        _apdefrepo = apdefrepo;
        _vgrepo = vgrepo;
        _slrepo = slrepo;
        _url = new UrlHelper();
        _trashid = _types.Load("SysRecycleBin").ID;
    }

    public ContentDetails AnalyzeContent(ContentReference contentRef, string lang=null)
    {
        var mastercnt = (lang == null) ? _repo.Get<IContent>(contentRef) : _repo.Get<IContent>(contentRef, new CultureInfo(lang));
        if (mastercnt == null)
        {
            return null;
        }
        if (mastercnt.ContentTypeID == _trashid) return null; //Skip if it's the recycle bin

        //Get ancestors
        var ancestors = _repo.GetAncestors(contentRef).Select(a => a.Name).Reverse().ToList();

        //Get versions
        var versions = (lang == null) ? _vrepo.List(contentRef) : _vrepo.List(contentRef, lang);

        //Which version should we use?
        var draft = (lang == null) ? versions.Where(cv => cv.IsMasterLanguageBranch).OrderByDescending(cv => cv.MasterVersionID).First() : versions.Where(cv => c
        var draftcnt = (lang == null) ? _repo.Get<IContent>(draft.ContentLink) : _repo.Get<IContent>(draft.ContentLink, new CultureInfo(lang));
        var refs = _repo.GetReferencesToContent(contentRef, false); //Does this work?
        var site = _resolver.GetByContent(contentRef, true);

        var apdef = _apdefrepo.ResolveAsync(contentRef).Result;
        var revs = apdef?.Definition?.Steps?.SelectMany(st => st.Reviewers).Select(r => r.Name).ToArray();
        string reviewers = (revs == null || apdef.Definition.IsEnabled == false) ? "(No Approval Flow)" : string.Join(",", revs);

        ContentDetails rt = new ContentDetails()
        {
            ContentId = contentRef.ToReferenceWithoutVersion().ToString(),
            Language = (mastercnt is ILocalizable) ? (mastercnt as ILocalizable).Language.TwoLetterISOLanguageName : "",
            ContentType = _types.Load(mastercnt.ContentTypeID).Name,
            MainType = (mastercnt is PageData) ? "Page" : (mastercnt is MediaData) ? "Media" : (mastercnt is BlockData) ? "Block" : "Other",
            Url = (mastercnt is PageData || mastercnt is MediaData) ? _url.ContentUrl(contentRef) : string.Empty,
            SiteName = site?.Name,
            Breadcrumb = string.Join(" > ", ancestors.ToArray()),
            Name = draftcnt.Name,
            DraftStatus = draft.Status.ToString(),
            CreatedBy = (draftcnt as IChangeTrackable)?.CreatedBy,
            CreatedDate = (draftcnt as IChangeTrackable)?.Created,
            ChangedBy = (draftcnt as IChangeTrackable)?.ChangedBy,
            ChangedDate = (draftcnt as IChangeTrackable)?.Changed,
            SavedBy = draft.SavedBy,
            SavedDate = draft.Saved,
            PublishedDate = (mastercnt is IVersionable) ? (mastercnt as IVersionable).StartPublish : null,
            StopPublishDate = (mastercnt is IVersionable) ? (mastercnt as IVersionable).StopPublish : null,
            Languages = (mastercnt is ILocalizable)? string.Join(", ", (mastercnt as ILocalizable).ExistingLanguages.Select(ci => ci.TwoLetterISOLanguageName).To
            UsesCount = refs.Select(r => r.ReferencedID.ToReferenceWithoutVersion().ToString()).Distinct().Count(),
            Uses = string.Join(", ", refs.Select(r => r.ReferencedID.ToReferenceWithoutVersion().ToString()).Distinct().ToArray()),
            Versions = versions.Count(),
            Reviewers  = reviewers
        };

        return rt;
    }

    protected IEnumerable<VisitorGroupUse> AnalyzeVisitorGroups(IContent c)
    {
        foreach (var p in c.Property)
        {
            if (p.Value == null) continue;
            if (p.PropertyValueType == typeof(ContentArea))
            {
                var ca = p.Value as ContentArea;
                if (ca == null) continue;
                foreach (var f in ca.Items.Where(i => i.AllowedRoles != null && i.AllowedRoles.Any()))
                {
                    //Match! This page uses the visitor groups in i.AllowedRoles. Record.
                    foreach (var r in f.AllowedRoles)
                    {
                        VisitorGroupUse vgu = new VisitorGroupUse();
                        vgu.VisitorGroup = r;
                        vgu.Content = c.ContentLink;
                        vgu.PropertyName = p.Name;
                        vgu.ContentName = (c as IContent).Name;
                        vgu.ContentType = (c is PageData) ? "Page" : (c is BlockData) ? "Block" : "Other";

                        yield return vgu;
                    }
                }
            }
            else if (p.PropertyValueType == typeof(XhtmlString))
            {
                var ca = p.Value as XhtmlString;
                if (ca == null) continue;
                foreach (var f in ca.Fragments.Where(fr => fr is EPiServer.Core.Html.StringParsing.PersonalizedContentFragment))
                {

                    var j = f as EPiServer.Core.Html.StringParsing.PersonalizedContentFragment;
                    var roles = j.GetRoles();
                    foreach (var r in roles)
                    {
                        VisitorGroupUse vgu = new VisitorGroupUse();
                        vgu.VisitorGroup = r;
                        vgu.Content = c.ContentLink;
                        vgu.PropertyName = p.Name;
                        vgu.ContentName = (c as IContent).Name;
```

```
                    vgu.ContentType = (c is PageData) ? "Page" : (c is BlockData) ? "Block" : "Other";
                    yield return vgu;
                }
            }

        }
    }
}


public void AnalyzeAllVisitorGroups(Action<string> StatusChanged, Func<bool> ShouldStop)
{
    VisitorGroupDetails = new List<VisitorGroupDetails>();
    var visitorgrouplist = _vgrepo.List().ToList();

    foreach (var vg in visitorgrouplist)
    {
        StatusChanged($"Analyzing visitor group {vg.Name}");
        foreach(var crt in vg.Criteria)
        {
            var vgd = new VisitorGroupDetails()
            {
                Name = vg.Name,
                Notes = vg.Notes,
                Operator = vg.CriteriaOperator.ToString(),
                IsSecurity = vg.IsSecurityRole,
                EnableStatistics = vg.EnableStatistics,
                PointsThreshold = vg.PointsThreshold,
                CriteriaType = crt.TypeName,
                CriteriaModel = JsonConvert.SerializeObject(crt.Model,Formatting.Indented),
                CriteriaPoints = crt.Points,
                CriteriaRequired = crt.Required,
                DefinitionName = crt.Definition.DisplayName
            };
            VisitorGroupDetails.Add(vgd);
        }


        if (ShouldStop()) break;
    }
}


public void AnalyzeAllContent(Action<string> StatusChanged, Func<bool> ShouldStop)
{
    ContentDetails = new List<ContentDetails>();
    VisitorGroupUses = new List<VisitorGroupUse>();

    Queue<ContentReference> todo = new Queue<ContentReference>();
    Dictionary<ContentReference, bool> done = new Dictionary<ContentReference, bool>();
    usedBy = new Dictionary<ContentReference, List<ContentReference>>();

    todo.Enqueue(ContentReference.GlobalBlockFolder);
    todo.Enqueue(ContentReference.SiteBlockFolder);
    _sites.List().ToList().ForEach(sd => {
        todo.Enqueue(sd.ContentAssetsRoot);
        todo.Enqueue(sd.GlobalAssetsRoot);
        todo.Enqueue(sd.SiteAssetsRoot);
        todo.Enqueue(sd.StartPage);
    });
    while (todo.Count > 0)
    {
        var cr = todo.Dequeue();
        if (done.ContainsKey(cr)) continue;
        done.Add(cr, true);
        var cnt = _repo.Get<IContent>(cr);
        if (cnt == null) continue;

        if (cnt is ILocalizable && (cnt as ILocalizable).ExistingLanguages.Count() > 1)
        {
            var langs = (cnt as ILocalizable).ExistingLanguages.Select(ci => ci.TwoLetterISOLanguageName);
            foreach (var l in langs.Where(ll => ll != (cnt as ILocalizable).Language.TwoLetterISOLanguageName))
            {
                //Analyze other language versions of content
                var langcntdetails = AnalyzeContent(cr, l);
                if (langcntdetails != null) ContentDetails.Add(langcntdetails);
            }
        }

        var cntdetails = AnalyzeContent(cr);
        if (cntdetails != null)
        {
            ContentDetails.Add(cntdetails);
            var refs = _repo.GetReferencesToContent(cr, false);
            usedBy.Add(cr.ToReferenceWithoutVersion(), _slrepo.Load(cr.ToReferenceWithoutVersion(), true).Select(sl => sl.OwnerContentLink.ToReferenceWithou
            VisitorGroupUses.AddRange(AnalyzeVisitorGroups(cnt));

        }
        var children = _repo.GetChildren<IContent>(cr);
        children.ToList().ForEach(c => todo.Enqueue(c.ContentLink));
        if (ShouldStop()) return;
        StatusChanged($"Analyzing Content. In progress, done with {ContentDetails.Count} content items, {VisitorGroupUses.Count} usages of visitorgroups,
    }

    //Update Visitor Group Uses
    var visitorgrouplist = _vgrepo.List().ToList();
    var vgcount = VisitorGroupUses.Count;
    for(int i = 0; i < vgcount; i++)
    {
        var vgu = VisitorGroupUses[i];
        var vg = visitorgrouplist.Where(g => g.Id.ToString() == vgu.VisitorGroup).FirstOrDefault();
        vgu.VisitorGroup = vg.Name;
        if (vgu.ContentType == "Page")
        {
            vgu.PageId = vgu.Content.ToReferenceWithoutVersion().ToString();
            vgu.PageName = vgu.ContentName;

        } else
        {
            var pages = new List<PageData>();
            var lst = usedBy[vgu.Content];
            while (lst.Any())
            {
                var itm = _repo.Get<IContent>(lst.First());
                lst.RemoveAt(0);
                if (itm is PageData) pages.Add((PageData)itm);
                else if (usedBy.ContainsKey(itm.ContentLink.ToReferenceWithoutVersion()))
                {
                    lst.AddRange(usedBy[itm.ContentLink.ToReferenceWithoutVersion()]);
                }
            }
            VisitorGroupUses.AddRange(pages.Select(p => new VisitorGroupUse()
            {
```

```
                    VisitorGroup = vgu.VisitorGroup,
                    Content = vgu.Content,
                    PropertyName = vgu.PropertyName,
                    ContentName = vgu.ContentName,
                    ContentType = vgu.ContentType,
                    PageId = p.ContentLink.ToReferenceWithoutVersion().ToString(),
                    PageName = p.Name
                }));
            }

        }
        // List visitor groups that personalize each content item
        ContentDetails.ForEach(cd =>
        {
            cd.VisitorGroupsUsed = String.Join(",", VisitorGroupUses.Where(vgu => vgu.PageId == cd.ContentId || vgu.Content.ToReferenceWithoutVersion().ToString
        });

    }


}

public class ContentDetails
{
    [ReportColumn(Title = "Content ID", Order = 10)]
    public string ContentId { get; set; }
    [ReportColumn(Title = "Language", Order = 15)]
    public string Language { get; set; }
    [ReportColumn(Title = "Content Type", Order = 20)]
    public string ContentType { get; set; }
    [ReportColumn(Title = "Main Type", Order = 30)]
    public string MainType { get; set; }
    [ReportColumn(Title = "Url", Order = 40)]
    public string Url { get; set; }
    [ReportColumn(Title = "Site Name", Order = 50)]
    public string SiteName { get; set; }
    [ReportColumn(Title = "Breadcrumb", Order = 60)]
    public string Breadcrumb { get; set; }
    [ReportColumn(Title = "Name", Order = 70)]
    public string Name { get; set; }
    [ReportColumn(Title = "DraftStatus", Order = 80)]
    public string DraftStatus { get; set; }

    [ReportColumn(Title = "Created By", Order = 90)]
    public string CreatedBy { get; set; }

    [ReportColumn(Title = "Created", Order = 100, FormatString = "yyyy-MM-dd HH:mm")]
    public DateTime? CreatedDate { get; set; }

    [ReportColumn(Title = "Changed By", Order = 110)]
    public string ChangedBy { get; set; }
    [ReportColumn(Title = "Changed", Order = 120, FormatString = "yyyy-MM-dd HH:mm")]
    public DateTime? ChangedDate { get; set; }
    [ReportColumn(Title = "Saved By", Order = 130)]
    public string SavedBy { get; set; }
    [ReportColumn(Title = "Saved", Order = 140, FormatString = "yyyy-MM-dd HH:mm")]
    public DateTime? SavedDate { get; set; }

    [ReportColumn(Title = "Published", Order = 150, FormatString = "yyyy-MM-dd HH:mm")]
    public DateTime? PublishedDate { get; set; }
    [ReportColumn(Title = "Published until", Order = 160, FormatString = "yyyy-MM-dd HH:mm", DefaultValue ="")]
    public DateTime? StopPublishDate{ get; set; }
    [ReportColumn(Title = "Master Language", Order = 170)]
    public string MasterLanguage { get; set; }
    [ReportColumn(Title = "Languages", Order = 180)]
    public string Languages { get; set; }
    [ReportColumn(Title = "How many places is it used", Order = 190)]
    public int UsesCount { get; set; }
    [ReportColumn(Title = "Where is it used", Order = 200)]
    public string Uses { get; set; }
    [ReportColumn(Title = "How many versions", Order = 210)]
    public int Versions { get; set; }
    [ReportColumn(Title = "List of reviewers", Order = 220)]
    public string Reviewers { get; set; }

    [ReportColumn(Title = "Visitor Groups used", Order = 230)]
    public string VisitorGroupsUsed { get; set; }

}

public class VisitorGroupDetails
{

    [ReportColumn(Title = "Visitor Group Name", Order = 10)]
    public string Name { get; set; }

    [ReportColumn(Title = "Visitor Group Notes", Order = 20)]
    public string Notes { get; set; }

    [ReportColumn(Title = "Criteria Operator", Order = 30)]
    public string Operator { get; set; }

    [ReportColumn(Title = "Is Security Role", Order = 40)]
    public bool IsSecurity { get; set; }

    [ReportColumn(Title = "Enable statistics", Order = 50)]
    public bool EnableStatistics { get; set; }

    [ReportColumn(Title = "Points Threshold", Order = 60)]
    public int PointsThreshold { get; set; }

    [ReportColumn(Title = "Criteria Name", Order = 70)]
    public string DefinitionName { get; set; }

    [ReportColumn(Title = "Criteria Type", Order = 80)]
    public string CriteriaType { get; set; }

    [ReportColumn(Title = "Criteria Model", Order = 90)]
    public string CriteriaModel { get; set; }

    [ReportColumn(Title = "Criteria is Required", Order = 100)]
    public bool CriteriaRequired { get; set; }

    [ReportColumn(Title = "Criteria Points", Order = 110)]
    public int CriteriaPoints { get; set; }

}


//TODO: Content Type Use

//TODO: Language Use

```

```
    public class VisitorGroupUse
    {
        [ReportColumn(Title = "Page Id", Order = 10)]
        public string PageId { get; set; }

        [ReportColumn(Title = "Page Name", Order = 20)]
        public string PageName { get; set; }

        [ReportColumn(Title = "Visitor Group", Order = 30)]
        public string VisitorGroup { get; set; }

        [ReportColumn(Title = "Property Name", Order = 40)]
        public string PropertyName { get; set; }

        [ReportColumn(Title = "Content Name", Order = 60)]
        public string ContentName { get; set; }

        [ReportColumn(Title = "Content Type", Order = 70)]
        public string ContentType { get; set; }

        [ReportColumn(Title = "Content Id", Order = 80)]
        public ContentReference Content { get; set; }


    }

    /// <summary>
    /// Contains details on how this should be used in the report. Including the column title, order, formatstring
    /// </summary>
    [AttributeUsage(AttributeTargets.Property, AllowMultiple =false)]
    public class ReportColumnAttribute : Attribute {
        public string Title { get; set; }
        public int Order { get; set; }
        public string FormatString { get; set; }

        public string DefaultValue { get; set; }
    }

}
```

ContentAnalyzer.cs hosted with ♥ by GitHub view raw

```
using EPiServer;
using EPiServer.Core;
using EPiServer.DataAbstraction;
using EPiServer.Framework.Blobs;
using EPiServer.ServiceLocation;
using OfficeOpenXml;
using System;
using System.Collections.Generic;
using System.IO;
using System.Linq;

namespace CodeArt.Episerver.ContentReport
{
    //Requires EPPlusFree, Nuget command: Install-Package EPPlusFree
    public class ContentReport : IDisposable
    {
        private ExcelPackage package;
        private Injected<ContentMediaResolver> _mediaResolver;
        private Injected<IContentRepository> _repo;
        private Injected<IContentTypeRepository> _types;
        private Injected<IBlobFactory> _blobFactory;

        public ContentReport()
        {
            package = new ExcelPackage();
        }


        public void AddListWorksheet<G>(string SheetName, IEnumerable<G> Items)
        {
            ExcelWorksheet ws = package.Workbook.Worksheets.Add(SheetName);
            var type = typeof(G);
            var properties=type.GetProperties(System.Reflection.BindingFlags.Public | System.Reflection.BindingFlags.Instance | System.Reflection.BindingFlags.GetP
            var attributes = properties.Select(pi => new { property = pi, attribute = pi.GetCustomAttributes(typeof(ReportColumnAttribute), true).First() as ReportColu
            //Write header
            int column = 1;
            foreach(var attr in attributes)
            {
                ws.Cells[1, column].Value = attr.attribute.Title ?? attr.property.Name;
                ws.Cells[1, column].Style.Font.Bold = true;
                column++;
            }

            int row = 2;
            foreach(G itm in Items)
            {
                column = 1;
                foreach (var attr in attributes)
                {
                    //ws.Cells[row, column].Value= (attr.attribute.FormatString!=null)? attr.property.GetValue(itm).ToString()
                    object v = attr.property.GetValue(itm);
                    string cell = string.Empty;
                    if (v == null)
                    {
                        cell = attr.attribute.DefaultValue ?? string.Empty;
                    }
                    else if (v is DateTime?)
                    {
                        cell = ((DateTime?)v).Value.ToString(attr.attribute.FormatString ?? "yyyy-MM-dd HH:mm");
                    }
                    else
                    {
                        cell = v.ToString();
                    }
                    ws.Cells[row, column].Value = cell;
                    column++;
                }
                row++;
            }
            for (int i = 1; i < column; i++)
            {
                ws.Column(i).AutoFit();
            }

        }

        public void Save(string Filename, ContentReference Parent)
        {
            Type t = _mediaResolver.Service.GetFirstMatching(Path.GetExtension(Filename));
            var contentType=_types.Service.Load(t);
```

```
            var reportcnt = _repo.Service.GetChildren<MediaData>(Parent).Where(md => md.Name == Filename && md.ContentTypeID == contentType.ID).FirstOrDefau
            if (reportcnt != null)
            {
                reportcnt = (MediaData)reportcnt.CreateWritableClone();
            }
            else
            {
                reportcnt = _repo.Service.GetDefault<MediaData>(Parent, contentType.ID);
                reportcnt.Name = Filename;
            }

            var blob =_blobFactory.Service.CreateBlob(reportcnt.BinaryDataContainer, ".xlsx");
            using (var s = blob.OpenWrite())
            {
                BinaryWriter w = new BinaryWriter(s);
                w.Write(package.GetAsByteArray());
                w.Flush();
            }
            reportcnt.BinaryData = blob;
            _repo.Service.Save(reportcnt, EPiServer.DataAccess.SaveAction.Publish, EPiServer.Security.AccessLevel.NoAccess);

        }

        public void Dispose()
        {
            package.Dispose();
        }
    }


}
```

ContentReport.cs hosted with ♥ by GitHub view raw

```
using EPiServer;
using EPiServer.Core;
using EPiServer.PlugIn;
using EPiServer.Scheduler;
using EPiServer.ServiceLocation;
using System;
using System.Linq;

namespace CodeArt.Episerver.ContentReport
{
    [ScheduledPlugIn(DisplayName = "Content Report Job")]
    public class ContentReportJob : ScheduledJobBase
    {
        private bool _stopSignaled;

        private Injected<IContentRepository> _repo;

        public ContentReportJob()
        {
            IsStoppable = true;
        }

        /// <summary>
        /// Called when a user clicks on Stop for a manually started job, or when ASP.NET shuts down.
        /// </summary>
        public override void Stop()
        {
            _stopSignaled = true;
        }



        /// <summary>
        /// Called when a scheduled job executes
        /// </summary>
        /// <returns>A status message to be stored in the database log and visible from admin mode</returns>
        public override string Execute()
        {
            OnStatusChanged("Starting to build report");

            ContentAnalyzer contentAnalyzer = ServiceLocator.Current.GetInstance<ContentAnalyzer>();



            //Ensure output folder
            ContentReference outputRef = ContentReference.EmptyReference;
            ContentFolder outputfolder = _repo.Service.GetChildren<ContentFolder>(ContentReference.GlobalBlockFolder).Where(cf => cf.Name == "Internal Reports").Fir
            if (outputfolder == null)
            {
                var tmp = _repo.Service.GetDefault<ContentFolder>(ContentReference.GlobalBlockFolder);
                tmp.Name = "Internal Reports";
                //TODO: Access rights, only admin access!!
                outputRef = _repo.Service.Save(tmp, EPiServer.DataAccess.SaveAction.Publish, EPiServer.Security.AccessLevel.NoAccess);
            }
            else outputRef = outputfolder.ContentLink;

            //Analyze visitor groups
            contentAnalyzer.AnalyzeAllVisitorGroups(OnStatusChanged, () => _stopSignaled);

            //Analyze content
            contentAnalyzer.AnalyzeAllContent(OnStatusChanged, () => _stopSignaled);
            string repname = "Report " + DateTime.Now.ToString("yyyy-MM-dd HH:mm") + ".xlsx";

            using (var report = new ContentReport())
            {
                report.AddListWorksheet<ContentDetails>("Content Details", contentAnalyzer.ContentDetails.OrderBy(ob => ob.CreatedDate));
                report.AddListWorksheet<VisitorGroupDetails>("Visitor Group Details", contentAnalyzer.VisitorGroupDetails.OrderBy(ob => ob.Name));
                report.AddListWorksheet<VisitorGroupUse>("Visitor Group Usage", contentAnalyzer.VisitorGroupUses.OrderBy(ob => ob.VisitorGroup));
                report.Save(repname, outputRef);
            }

            return $"Job done. Analyzed {contentAnalyzer.ContentDetails.Count} content items and {contentAnalyzer.VisitorGroupDetails.Count} Visitor group details.
        }

    }



}
```

ContentReportJob.cs hosted with ♥ by GitHub view raw

CodeArt ApS
Teknikerbyen 5, 2830 Virum, Denmark
Email: info@codeart.dk
Phone: +45 26 13 66 96
CVR: 39680688