Sanjay Kumar Oct 6, 2025 307 (5 votes)

Automated Page Audit for Large Content Sites in Optimizely

Large content sites often face significant challenges in maintaining visibility and control over thousands of pages. Content managers struggle to track publishing status, identify outdated or unpublished content, and ensure overall content quality. Without automation, auditing becomes time-consuming, error-prone, and inefficient, creating risks in governance, compliance, and user experience.

Managing content at enterprise scale is difficult because:

- **Limited Visibility** – No consolidated view of all site content.
- **Publishing Confusion** – Hard to track what is published, unpublished, or pending.
- **Lifecycle Issues** – Identifying outdated or expired content requires manual checks.
- **Change Tracking** – Monitoring edits across thousands of pages is resource-intensive.
- **Data Exporting** – Manual CSV exports are slow, error-prone, and not scalable.
- **High Effort** – Auditing consumes substantial IT and administrative time.

Use Cases:

1. Scheduled Report Export:

Extend the reports feature by integrating a scheduled job that automatically exports selected page and content data (e.g., title, URL, author, last modified date) into a structured CSV or Excel format for content audit and review.

2. Metadata Validation and Updates

Identify and update missing or incomplete metadata (e.g., Title, Description, or Keywords) for published pages to maintain content accuracy, SEO compliance, and adherence to governance standards.

3. Expired or Archived Content

Detect and list archived or expired pages that are no longer active or relevant, enabling content teams to clean up and improve site performance and user experience.

4. Long-standing Draft Pages

Highlight draft pages that have been sitting unpublished for an extended period. This helps content managers review, update, or remove outdated drafts to ensure content freshness.

5. SEO-friendly Simplified URLs

Automate updates to simple addresses (SEO-friendly URLs) to ensure consistency with naming conventions and improve search visibility.

6. Approved but Unpublished Pages

Identify pages that have been approved by authors or editors but not yet published, allowing administrators to review and publish them efficiently.

Solution

A custom **PageReportsJob** automates page auditing for large content sites. It generates CSV reports that give administrators and content managers an overview of content structure, publishing status, and updates.

We solves these issues with an automated job process that:

- Automatically scans and identifies all content types and statuses.
- Generates standardized CSV reports for consistency.
- Efficiently processes large datasets using paging and async execution.
- Provides near real-time insights on content status and updates.
- Secures access with role-based permissions.

```
using EPiServer.Logging;
using EPiServer.PlugIn;
using EPiServer.Scheduler;
using EPiServer.Security;
using EPiServer.ServiceLocation;
using EPiServer.Web;
using Microsoft.Extensions.Options;
using Nito.AsyncEx;
using MyCompany.Web.Features.Reports.ExistingPages.Models;
using MyCompany.Web.Infrastructure.Cms;
using MyCompany.Web.Infrastructure.Cms.Settings;
using static MyCompany.Web.Features.Core.Constants.CommonConstants;

namespace MyCompany.Web.Features.Jobs
{
    [ScheduledPlugIn(
        DisplayName = "Page Report Job ",
        Description = "Exports existing pages to CSV for administrators and content managers",
        IntervalType = ScheduledIntervalType.None,
        SortIndex = -1004 + 'D')]
    public class PageReportJob : ScheduledJobBase
    {
        private readonly ILogger _logger;
        private readonly IMediaFileService _mediaFileService;
        private readonly IContentRepository _contentRepository;
        private readonly IContentReportQueryService _contentReportQueryService;
        private readonly IContentVersionRepository _contentVersionRepository;
        private readonly ISettingsService _settingsService;
        private readonly ExistingPagesReportOptions _options;

        private bool _stopSignaled;

        public PageReportJob (
            IMediaFileService mediaFileService,
            IContentRepository contentRepository,
            IContentReportQueryService contentReportQueryService,
            IContentVersionRepository contentVersionRepository,
            ISettingsService settingsService,
            IOptions<ExistingPagesReportOptions> options)
        {
            _mediaFileService = mediaFileService;
            _contentRepository = contentRepository;
            _contentReportQueryService = contentReportQueryService;
            _contentVersionRepository = contentVersionRepository;
            _settingsService = settingsService;
            _options = options?.Value ?? new ExistingPagesReportOptions();
            _logger = LogManager.GetLogger();
            IsStoppable = true;
        }

        public override void Stop() => _stopSignaled = true;

        public override string Execute()
        {
            try
            {
                return AsyncContext.Run(GenerateReports);
            }
            catch (Exception ex)
            {
                _logger.Error(ex.StackTrace);
                return "The job stopped due to an exception.";
            }
        }

        private async Task<string> GenerateReports()
        {
            var reportQueryTypes = Enum.GetValues(typeof(ContentReportQueryType))
                .Cast<ContentReportQueryType>()
                .Select(v => v.ToString())
                .ToList();
```

```
            // Adding a custom query type
            reportQueryTypes.Add("NotPublishedPages");

            foreach (var reportQueryType in reportQueryTypes)
            {
                if (_stopSignaled) break;
                await CreateReport(reportQueryType);
            }

            return "Exported all pages successfully.";
        }

        private async Task CreateReport(string reportQueryType)
        {
            try
            {
                _logger.Information($"Starting report creation for type '{reportQueryType}'.");

                var csvFolderPath = _mediaFileService.GetOrCreateFolder(
                    ExistingPageReportTypes.FolderName,
                    SiteDefinition.Current.SiteAssetsRoot);

                var pageCollection = await SetPagesByTypeOfQuery(reportQueryType);

                var pages = pageCollection
                    .Select(x => ExistingPageViewModel.Make(
                        x,
                        _contentVersionRepository.List(x.ContentLink).Where(v => v.LanguageBranch ==
x.Language?.Name)))
                    .Where(x => !string.IsNullOrWhiteSpace(x.Url))
                    .ToList();

                var fileName = $"{ExistingPageReportTypes.PrefixFileName}_{reportQueryType.ToLower()}";
                var media = _mediaFileService.SaveFile(
                    fileName,
                    ExistingPageReportTypes.FileExtension,
                    csvFolderPath,
                    new byte[0]);

                _mediaFileService.WriteCsv(media, pages);
                _contentRepository.Save(media, SaveAction.Publish, AccessLevel.NoAccess);

                _logger.Information($"Report '{fileName}' created and published with {pages.Count} rows.");
            }
            catch (Exception ex)
            {
                _logger.Error($"Error in CreateReport for type '{reportQueryType}': {ex}");
                throw;
            }
        }

        private async Task<PageDataCollection?> SetPagesByTypeOfQuery(string typeOfQuery)
        {
            var retryCounter = 0;
            do
            {
                try
                {
                    return await ReadPageDataCollection(typeOfQuery);
                }
                catch (Exception ex)
                {
                    retryCounter++;
                    _logger.Warning($"Error in SetPagesByTypeOfQuery (attempt {retryCounter}) for query
'{typeOfQuery}': {ex}");
                }
            }
            while (retryCounter < 3);

            _logger.Error($"Failed to get pages by type of query '{typeOfQuery}' after 3 attempts.");
            return null;
```

```
        }

        private async Task<PageDataCollection> ReadPageDataCollection(string typeOfQuery)
        {
            const int MaximumRows = 50;
            int receivedRecords, pageNumber = 0;

            var queryType = string.Equals("NotPublishedPages", typeOfQuery, StringComparison.OrdinalIgnoreCase)
                ? "ReadyToPublish"
                : typeOfQuery;

            var typeEnum = GetType(queryType);

            var startDate = (typeEnum == ContentReportQueryType.Changed && _options?.ChangedWindowDays is int
days && days > 0)
                ? DateTime.Now.AddDays(-days)
                : new DateTime(1900, 1, 1);

            if (typeEnum == ContentReportQueryType.Changed)
            {
                _logger.Information($"Using ChangedWindowDays of {_options.ChangedWindowDays}, start date
{startDate:yyyy-MM-dd}");
            }

            var query = new ContentReportQuery
            {
                Root = ContentReference.StartPage,
                StartDate = startDate,
                EndDate = DateTime.Now,
                PageSize = 1,
                PageNumber = pageNumber,
                TypeOfQuery = typeEnum,
                IsReadyToPublish = string.Equals("ReadyToPublish", typeOfQuery,
StringComparison.OrdinalIgnoreCase)
            };

            var totalRecords = await Task.Run(() =>
            {
                _contentReportQueryService.Get(query, out int outRows);
                return outRows;
            });

            var pageDataCollection = new PageDataCollection(totalRecords);

            do
            {
                query.PageNumber = pageNumber;
                query.PageSize = MaximumRows;

                var enumerable = await Task.Run(() => _contentReportQueryService.Get(query, out int _));
                foreach (ContentReference item in enumerable)
                {
                    if (_contentRepository.Get<IContent>(item) is PageData page)
                    {
                        pageDataCollection.Add(page);
                    }
                }

                pageNumber++;
                receivedRecords = MaximumRows * pageNumber + 1;
            }
            while (totalRecords > receivedRecords);

            _logger.Information($"Retrieved {pageDataCollection.Count} pages for query type '{typeOfQuery}'.");

            var filtered = ApplyGuards(pageDataCollection, typeEnum, typeOfQuery);
            return new PageDataCollection(filtered);
        }

        private IEnumerable<PageData> ApplyGuards(IEnumerable<PageData> items, ContentReportQueryType type,
string typeOfQuery)
        {
```

```
            var now = DateTime.Now;

            return type switch
            {
                ContentReportQueryType.Published => items.Where(p =>
                    p.CheckPublishedStatus(PagePublishedStatus.Published) &&
                    p.StartPublish <= now &&
                    (!p.StopPublish.HasValue || p.StopPublish > now)),

                ContentReportQueryType.ReadyToPublish when
                    string.Equals(typeOfQuery, "NotPublishedPages", StringComparison.OrdinalIgnoreCase) =>
                    items.Where(p => (!p.StopPublish.HasValue || p.StopPublish > now) &&
(!p.StartPublish.HasValue || p.StartPublish > now)),

                ContentReportQueryType.ReadyToPublish => items,
                ContentReportQueryType.Expired => items.Where(p => p.StopPublish.HasValue && p.StopPublish.Value
<= now),
                ContentReportQueryType.Changed => items,
                ContentReportQueryType.SimpleAddress => items.Where(p =>
!string.IsNullOrWhiteSpace(p.URLSegment)),

                _ => items
            };
        }

        private ContentReportQueryType GetType(string type) =>
            type switch
            {
                "Published" => ContentReportQueryType.Published,
                "ReadyToPublish" => ContentReportQueryType.ReadyToPublish,
                "Expired" => ContentReportQueryType.Expired,
                "Changed" => ContentReportQueryType.Changed,
                "SimpleAddress" => ContentReportQueryType.SimpleAddress,
                _ => ContentReportQueryType.Published
            };
    }
}
```

The system automatically generates six types of page reports: Published, Not Published, Ready to Publish, Expired, Changed, and Simple Address Pages.

Example: Page Audit Report (CSV Output)

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
1	Page Id	Page Title	Page Type	Page Url	Taxonomy - Audience	Taxonomy - Tag	Taxonomy -Category	Audience	Language	Published Date	Last Published By	Meta Title	Meta Description	Meta Author	Canonical Link
2	40305	2025 Event One	EventPage	https://www.optimizely.com/	Meet Us	A,B	Advancement	Community	en	25-09-25 14:26					
3	40303	2026 Event One	EventPage	https://www.optimizely.com/	Meet Us	A,B	Facilities	Community	en	25-09-25 14:32					
4	40304	2027 Event One	EventPage	https://www.optimizely.com/	Meet Us	A,B	Connected	Community	en	25-09-25 14:28					
5	40307	2028 Event One	EventPage	https://www.optimizely.com/	Meet Us	A,B	Advancement	Community	en-AU	25-09-25 14:17					
6	40399	About Us	AboutUsPage	https://www.optimizely.com/	Meet Us	A,B	Connected	Community	en	25-09-25 14:10					

Conclusion:
The **PagesAuditJob** represents a robust, enterprise-grade solution for content auditing and reporting. Its combination of scheduled automation, web interface accessibility, and comprehensive reporting capabilities makes it an essential tool for maintaining content quality and governance standards in Optimizely content management system.

Hope this post helps you strengthen and streamline your page audit process.

Cheers! 🥂

Oct 06, 2025

Comments

Ashish Rasal Oct 8, 2025 01:52 PM

Nice and helpful.

Please login to comment.

Latest blogs

[How to Add Multiple Authentication Providers to an Optimizely CMS 12 Site (Entra ID, Google, Facebook, and Local Identity)]
Modern websites often need to let users sign in with their corporate account (Entra ID), their social identity (Google, Facebook), or a simple...
Francisco Quintanilla | Oct 22, 2025 |

[Connecting the Dots Between Research and Specification to Implementation using NotebookLM]
Overview As part of my day to day role as a solution architect I overlap with many clients, partners, solutions and technologies. I am often...
Scott Reed | Oct 22, 2025

[MimeKit Vulnerability and EPiServer.CMS.Core Dependency Update]
Hi everyone, We want to inform you about a critical security vulnerability affecting older versions of the EPiServer.CMS.Core package due to its...
Bien Nguyen | Oct 21, 2025

[Speeding Up Local Development with a Fake OpenID Authentication Handler]
When working with OpenID authentication, local development often grinds to a halt waiting for identity servers, clients, and users to be configured...
Eric Herlitz | Oct 20, 2025 |

See all blogs