

MIGRATION APPROACH & TOOLKIT GUIDE

Oracle WebCenter Sites to Adobe Experience Manager

Content migration with the **wcs2aem** toolkit: export, mapping, transformation, packaging, DAM upload, validation and cutover

Prepared by	Onwardpath (Ganesh Narasimhan)
Date	23 October 2026
Version	0.1 (draft)
Toolkit	<code>wcs2aem</code> 0.1.0, Python 3.9 or later, standard library only
Targets	AEM 6.5 and AEM as a Cloud Service
Source	<code>E:\onwardpath\wcs-aem-migration</code> . The full mapping reference is in <code>README.md</code> ; <code>samples/mapping.json</code> is a complete worked example.
Status	Toolkit built and tested offline. It has not yet been run against a live WCS instance or AEM author.

Contents

1. Executive summary	8. DAM binary strategy
2. Scope	9. URLs, redirects and SEO
3. Solution architecture	10. Validation and reporting
4. Content model mapping	11. Migration runbook
5. Pipeline stages	12. Risks and open items
6. Mapping configuration	13. Appendix A: CLI reference
7. Rich text and link handling	14. Appendix B: Sample run results

1. Executive summary

This document describes how content moves from Oracle WebCenter Sites (WCS) into Adobe Experience Manager (AEM). It also explains how to use **wcs2aem**, the toolkit built to automate that move. The toolkit reads assets from the WCS REST API and converts them to a neutral intermediate format. It then applies a declarative JSON mapping that turns each WCS asset type into one of three things:

- an AEM **page** built from components,
- a structured **Content Fragment**,
- a **DAM asset**.

The result is a standard FileVault content package that installs through CRX Package Manager. Each run also produces a migration report and a redirect map.

6 CLI commands, from export to upload	0 third-party dependencies	23 automated tests, all passing	2 WCS REST shapes supported
--	--------------------------------------	--	--

The approach is built for repeated runs. Each run is idempotent and the mapping file is version-controlled. The report lists every type that has no mapping rule, every broken reference and every link that could not be rewritten. The team can therefore converge on a clean migration through trial runs, then do a final delta run at content freeze.

Current status. Every stage has been exercised end to end on a synthetic healthcare site that mirrors typical WCS structures. The exact REST payload shapes of the client's WCS version, and installation on a real AEM author, still need to be verified (see section 12).

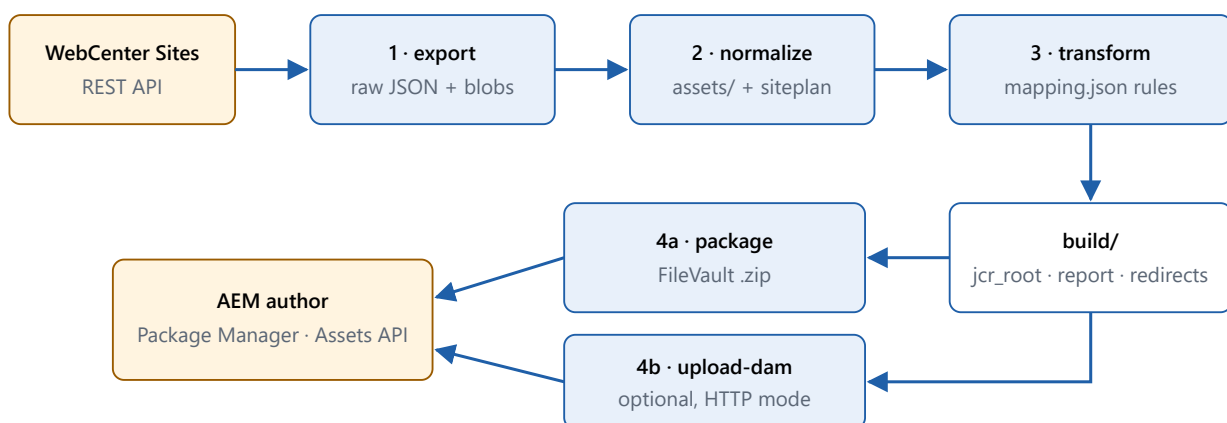
2. Scope

In scope (automated)	Out of scope (manual or separate workflow)
- Page assets and the site plan, which become the AEM page hierarchy - Flex and basic content assets, which become Content Fragments or page components - Image and document blobs, which become DAM assets with metadata - Attribute values: string, number, date, list, asset reference, blob - Named associations (for example related content) - Rich text: embedded links, inclusions and Satellite URLs - Vanity URLs (webreferences) and legacy Satellite URLs, which become redirects - Traceability: the source asset id is written on every node	- Templates, CSElements and SiteEntries. These are code, rebuilt as AEM components and editable templates. - Multilingual content: Dimension and DimensionSet to AEM language copies and MSM - WCS Engage segments, recommendations and personalisation - Workflow, approval and publish state - ACLs, users and groups - BlobServer image URLs embedded directly in rich text (reported, not resolved) - Front-end design, components, dispatcher and CDN configuration

Asset types that are out of scope are not silently lost. Each run either lists them in the `skip` list or reports them as *unmapped*.

3. Solution architecture

The pipeline has four stages. Each stage writes its output to disk, so you can inspect, diff and re-run any stage on its own.



Design principles

- **Declarative mapping.** All site-specific decisions live in `mapping.json`, not in code. Business analysts can review the file and it is versioned with the project.
- **Two-pass transform.** Every asset's AEM path is planned before any node is built, so references and links resolve regardless of order.
- **Source isolation.** Only `normalize.py` knows the WCS payload shapes. CSDT exports or database extracts can feed the same pipeline by writing the normalized format.

- **Nothing silent.** Every gap is written to the report: unmapped types, missing references, unresolved links, orphan pages and missing binaries.
- **No secrets on disk.** Credentials are read only from environment variables.

4. Content model mapping

The table shows how typical WCS constructs land in AEM. The exact mapping for each asset type and subtype is set per project in `mapping.json`.

WebCenter Sites	AEM	Notes
Site	Site root page under <code>/content/<site>/<country>/<lang></code>	Set by <code>aem.contentRoot</code> . The WCS home page becomes this page.
Site plan (SitePlanTree)	Page hierarchy and sibling order	Taken from the REST navigation endpoint, or from a CSV extract of SitePlanTree.
Page asset (by subtype)	<code>cq:Page</code> with an editable template	The subtype selects the template, the page resource type and the component layout.
Page attributes	<code>jcr:content</code> properties and components	Title, description and SEO fields go to page properties. Body content goes to Text, Image or Teaser components.
Flex/basic content asset (for example Article or Doctor)	Structured Content Fragment	The CF model must exist under <code>/conf</code> . Elements are mapped field by field.
Content shown only on a page	Components on that page (<code>inline</code>)	Created one per referenced item with <code>foreach</code> .
Image and document assets (blob attribute)	<code>dam:Asset</code> under <code>/content/dam/<site></code>	Metadata goes to <code>jcr:content/metadata</code> (for example <code>dc:title</code>).
Asset reference attribute	Path property (<code>fileReference</code> , <code>linkURL</code> , CF reference)	Resolved through the planned path map. Missing targets are reported.
Named association	List component, CF multi-reference or <code>foreach</code> components	Read with the <code>assoc:<name></code> source.
Webreference (vanity URL)	Redirect entry	Written to <code>redirects.csv</code> and <code>redirectmap.txt</code> .
Template, CSElement, SiteEntry	AEM components, templates and HTL (rebuilt)	Code, not content. Listed in <code>skip</code> .
Dimension and translations	Language copies and MSM	Not yet automated (see section 12).

5. Pipeline stages

5.1 Export

`export` pages through each configured asset type with the WCS search resource. For each asset it fetches the full payload, downloads every blob and saves the raw JSON exactly as returned. It then fetches the site navigation to build `siteplan.json`.

- **API styles.** Two are supported: `classic` (`/REST/sites/{site}/types/{type}/...`, 11g and 12c) and `v1` (`/REST/resources/v1/...`, 12c).
- **Authentication.** A CAS multiticket (the WCS default) or HTTP Basic. An expired CAS ticket is refreshed automatically.
- **Resumable.** Assets already on disk are skipped unless you pass `--force`. Failures are logged in `export-log.json`.
- **Throttling.** `delaySeconds` protects a production WCS during business hours.

5.2 Normalize

Raw payloads are converted to a single normalized asset format. This format is the contract between any source and the transform stage.

```
{ "id": "Page:1330000000004", "type": "Page", "subtype": "NV_ServicePage", "name": "Cardiology",
  "attributes": { "title": "Cardiology",
    "heroImage": { "$asset": "NV_Image:13300000000102"},
    "brochure": { "$blob": { "file": "...", "filename": "x.pdf", "mimetype":
"application/pdf" } } },
  "associations": { "related": [ { "$asset": "Page:1330000000005" } ] },
  "webreferences": [ "/services/cardiology" ] }
```

5.3 Transform

The transform runs in two passes. **Plan** assigns an AEM path to every mapped asset, using site-plan order and collision-safe slugs, with orphan pages placed in a holding folder. **Build** then creates the JCR nodes, evaluates the mapping expressions, rewrites rich text and writes the FileVault tree to `build/jcr_root`.

5.4 Package and upload

`package` zips the build into a content package. It writes `filter.xml` from the roots actually produced (default mode `update`) and `properties.xml` with the name, group and version. If binaries are uploaded over HTTP, `upload-dam` pushes them to the same planned DAM paths, so page references stay valid (see section 8).

6. Mapping configuration

A rule is chosen by `Type/Subtype` first, then by `Type`. Each rule names a *target*.

Target	Creates	Key settings
<code>page</code>	<code>cq:Page</code>	<code>template</code> , <code>resourceType</code> , <code>properties</code> , <code>components[]</code> , <code>name</code> , <code>parent</code>
<code>contentFragment</code>	Structured CF (<code>dam:Asset</code>)	<code>model</code> , <code>folder</code> , <code>title</code> , <code>elements</code> , <code>metadata</code>
<code>damAsset</code>	DAM asset with original rendition	<code>binary</code> , <code>folder</code> , <code>name</code> , <code>metadata</code>
<code>inline</code>	Nothing on its own	Used through a page component's <code>foreach</code>
<code>skip</code>	Nothing	Counted in the report as skipped

Example: a page rule

```
"Page/NV_ServicePage": {
  "target": "page",
  "template": "/conf/novena/settings/wcm/templates/content-page",
  "resourceType": "novena/components/page",
  "properties": { "jcr:title": "attr:title", "jcr:description": "attr:summary" },
  "components": [
    { "name": "image", "resourceType": "novena/components/image", "when": "attr:heroImage",
      "properties": { "fileReference": "attr:heroImage|ref" } },
    { "name": "text", "resourceType": "novena/components/text",
      "properties": { "text": "attr:body|richtext", "textIsRich": "const:true|bool" } } ] }
```

Expressions

Every value in a rule is an expression of the form `source|filter|filter:arg`.

Sources	Meaning
<code>attr:<name></code>	An attribute value
<code>field:<name></code>	A core field: <code>name</code> , <code>description</code> , <code>id</code> , <code>cid</code> , <code>subtype</code> , <code>created</code> , <code>updated</code> or <code>template</code>
<code>assoc:<name></code>	The assets in a named association
<code>const:<value></code>	A literal value
<code>self</code> / <code>webref</code>	A reference to the current asset, or its first vanity URL
<code>page.<source></code>	Inside <code>foreach</code> , evaluates against the page rather than the item

Filters	Effect
<code>ref</code> , <code>link</code> , <code>title</code>	Reference to AEM path, to URL (adds <code>.html</code> for pages), or to the referenced asset's name
<code>richtext</code> , <code>text</code>	Rewrite WCS markup for AEM, or strip tags
<code>date</code> , <code>bool</code> , <code>long</code> , <code>double</code>	Convert to the JCR type
<code>slug</code> , <code>lower</code> , <code>upper</code> , <code>strip</code> , <code>truncate:n</code> , <code>prefix:x</code> , <code>suffix:x</code>	String helpers
<code>first</code> , <code>join:sep</code> , <code>split:sep</code> , <code>default:x</code>	List and fallback helpers
<code>filename</code> , <code>mimetype</code>	Properties of a blob

7. Rich text and link handling

WCS stores internal links and embedded assets in CKEditor markup that is meaningless outside WCS. The `richtext` filter rewrites it using the planned path map.

WCS markup	Result in AEM
------------	---------------

<code></code>	<code></code>
<code><a _cseembedtype="internal" _cid="..."_c="..."></code>	The same. WCS marker attributes are removed.
<code></code>	<code></code> for images, otherwise a link
<code>/cs/Satellite?c=Page&cid=...</code>	The new page URL
Link to a document asset	The DAM path of the document
<code></code>	Left in place and reported . The blob id cannot be traced back to an asset.

Links whose target was not migrated stay unchanged and are listed in the report as `unresolved_link`. Editors can then fix them in AEM, or the source can be corrected before the next run.

8. DAM binary strategy

Mode	How it works	When to use
<code>damMode:</code> "package" (default)	Binaries are stored in the content package as the asset's original rendition.	AEM 6.5 and small or medium asset volumes. Simple, with one artifact to install.
<code>damMode: "http"</code>	Binaries are left out of the package and listed in <code>dam-manifest.json</code> . <code>upload-dam</code> sends them through the Assets HTTP API (6.5) or direct binary upload (Cloud Service). Metadata is then set with a PUT.	AEM as a Cloud Service and large asset volumes. On Cloud Service, assets installed by package are not processed by Asset Compute and would need reprocessing.

In HTTP mode, upload the binaries **before** installing the content package, so that image and document references in pages and fragments resolve as soon as the content is installed.

9. URLs, redirects and SEO

Each run writes `redirects.csv` (old, new, asset) and `redirectmap.txt` (Apache RewriteMap format) with entries from three sources:

- webreferences (vanity URLs) of every migrated page and asset,
- legacy Satellite URL patterns set in `redirects.patterns` (placeholders `{type}`, `{cid}`, `{site}`, `{name}`, `{slug}`),
- new public URLs, computed with `aem.publicUrlMap` to match the dispatcher and Sling mapping (for example `/content/novena/us/en` becomes `/`).

Satellite URLs carry their identity in the query string. Redirecting them at the dispatcher therefore needs a rule that matches on `QUERY_STRING`, or a small redirect servlet. Plain path redirects can be loaded into a RewriteMap or ACS Commons Redirect Manager.

10. Validation and reporting

Every transform writes `report.md` and `report.json`. Running with `--strict` returns a non-zero exit code when anything is unmapped or has issues, so CI can gate on a clean run.

Issue kind	Meaning and typical fix
<code>unmapped</code>	An asset type or subtype has no rule. Add a rule, or add it to <code>skip</code> .
<code>missing_ref</code>	A referenced asset was not exported or not mapped. Export that type, or accept the gap.
<code>unresolved_link</code> / <code>unresolved_inclusion</code>	A rich-text link points at content that is not migrated. Fix it in the source or in AEM.
<code>unresolved_blobserver</code>	A BlobServer URL is embedded in rich text. Replace it with a proper asset embed.
<code>orphan_page</code>	A page is not placed in the site plan. It is put under the orphans folder, or skipped.
<code>missing_binary</code>	A blob failed to download or is empty. Re-export it with <code>--force</code> .
<code>home_not_found</code>	The configured <code>homePage</code> id is not in the export.

Validation after installation should include:

- comparing page and asset counts with the report,
- spot-checking pages in author and publish,
- a link check on the publish tier,
- replaying a sample of `redirects.csv` against the new site.

`path-map.json` (WCS id to AEM path) and the `wcsSourceId` property on every node let you audit any item in AEM back to its source.

11. Migration runbook

#	Phase	Activities	Exit criteria
1	Discovery	• Inventory asset types, subtypes and volumes. • Run <code>probe</code> against WCS to confirm the payload shapes. • Extract the site plan. • Identify the vanity URLs and legacy URL patterns.	Type inventory signed off; REST access confirmed
2	AEM foundation	• Build the components, editable templates and Content Fragment models. • Deploy <code>ui.apps</code> and <code>ui.config</code>. • Agree the content root and DAM folder structure.	Templates and models available on target author
3	Mapping	• Write <code>mapping.json</code> type by type. • Review it with content owners.	Every type is mapped or explicitly skipped

4	Trial runs	- Do a full export to a dev or QA author. - Work through the report until <code>--strict</code> passes, or the remaining issues are accepted. - Run UAT on the migrated pages.	Clean report; UAT sign-off
5	Content freeze and final run	- Freeze WCS editing. - Re-export with <code>--force</code>, transform and bump the package version. - Upload DAM (HTTP mode), install the package and publish.	Counts match the report; smoke tests pass
6	Cutover	- Load the redirects into the dispatcher or CDN. - Switch DNS. - Monitor 404s and add any missing redirects.	Traffic on AEM; 404 rate within threshold
7	Decommission	- Archive the raw export and <code>path-map.json</code>. - Retire WCS.	Archive verified

Keep the export, mapping and build directories for every run. Diffing two `jcr_root` trees is the quickest way to review what a mapping change actually did.

12. Risks and open items

Item	Severity	Mitigation
WCS REST payload shapes differ between versions and patch levels	High	Run <code>probe</code> first. Only <code>normalize.py</code> needs changing.
Package not yet installed on a real AEM author	High	Install the sample package on a local 6.5 or Cloud Service SDK author before the first trial run.
Direct binary upload (Cloud Service) and the SitePlanTree SQL are written from documentation but not yet run	Medium	Verify on a Cloud Service dev environment and against the client's database schema.
Multilingual sites (Dimension and translations)	Medium	Extend the mapping with per-locale content roots and set up language copies and MSM.
BlobServer images embedded in rich text	Medium	Map blob ids to assets from the database, or clean them up editorially.
Large asset volumes in one package	Medium	Use <code>damMode: "http"</code> and split packages by type or folder.
Content edited in WCS during the project	Low	Do a delta re-export at content freeze. Runs are idempotent.

Appendix A: CLI reference

Credentials come from the environment: `WCS_USER` and `WCS_PASSWORD`, then either `AEM_USER` and `AEM_PASSWORD` or `AEM_TOKEN`.

Command	Purpose and main options
probe	Fetches a few assets of one type and prints the raw and normalized JSON. Options: <code>--config</code> , <code>--type</code> , <code>--id</code>
export	Exports assets, blobs and the site plan from WCS. Options: <code>--config</code> , <code>--out</code> , <code>--types</code> , <code>--no-blobs</code> , <code>--force</code>
normalize	Rebuilds <code>assets/</code> from <code>raw/</code> offline. Options: <code>--export</code> , <code>--site</code>
transform	Turns an export into an AEM build directory. Options: <code>--mapping</code> , <code>--export</code> , <code>--out</code> , <code>--strict</code>
package	Zips a build into a content package. Options: <code>--build</code> , <code>--out</code> , <code>--name</code> , <code>--group</code> , <code>--version</code>
run	Runs <code>transform</code> and <code>package</code> in one step.
upload-dam	Uploads <code>dam-manifest.json</code> binaries. Options: <code>--build</code> , <code>--aem</code> , <code>--api assets-http direct-binary</code> , <code>--replace</code> , <code>--dry-run</code>

```
python -m wcs2aem probe      --config config.json --type Page
python -m wcs2aem export    --config config.json --out work/export
python -m wcs2aem transform --mapping mapping.json --export work/export --out work/build --strict
python -m wcs2aem package   --build work/build
python -m wcs2aem upload-dam --build work/build --aem https://author-pXXX-eYYY.adobecloud.com --api direct-binary
```

Appendix B: Sample run results

The bundled sample is a Novena-style healthcare site with 18 WCS assets. It deliberately includes the problems listed below, so you can see how the report shows them. Running the full pipeline on it produced:

Output	Result
Pages	8 (home, about, services with 2 child pages, doctors, contact, 1 orphan)
Content Fragments	2 doctor profiles, with rich-text bio and a photo reference
DAM assets	6 (5 images and 1 PDF brochure), each with metadata
Redirects	26 (vanity URLs and two Satellite URL patterns)
Unmapped / skipped	<code>NV_Promo</code> ×1 unmapped; <code>Template</code> ×1 skipped
Issues reported	1 orphan page, 1 missing image reference, 1 dead Satellite link, 1 BlobServer image
Rich text	Internal links, an image inclusion, a document link and a Satellite link were all rewritten correctly
Package	Well-formed FileVault zip. Every XML file parses; filter roots are <code>/content/novena/us/en</code> and <code>/content/dam/novena</code> .
Test suite	23 tests pass: serialization, rich text, both payload shapes, end-to-end pipeline, and the exporter against a fake WCS server