

Adaptadocx Documentation

0.1.0

Contents

Adaptadocx is a documentation-publishing system built around Antora. The platform assembles multilingual AsciiDoc sources and produces three outputs: **HTML**, **PDF** and **DOCX**. Built-in QA checks, security audits, and CI workflows ensure consistent quality and reproducible builds.

The system currently supports:

* Two locales (English and Russian)
* Custom Antora UI theming
* SVG diagram rendering
* Containerised builds with Docker
* Automated QA, security, and linting workflows
* Multiversion site by Git tags with versioned downloads

# Architecture Overview

Adaptadocx follows a modular design: Antora is the core generator, while auxiliary components extend its capabilities.

| Component | Purpose |
| --- | --- |
| Documentation Sources | AsciiDoc content for EN and RU in docs/ |
| Configuration | PDF theme and per-locale metadata in config/ |
| UI Components | Antora UI customisations in custom-ui/ |
| Build Scripts | Makefile and alternative Python orchestrator build.py |
| CI/CD | GitHub Actions workflows in .github/ |

# Core Features

## Multi-format Output

A single set of AsciiDoc files yields three independent artefacts.

* **HTML** — static site with a custom UI bundle and full-text search
* **PDF** — print/offline version with a custom theme and DejaVu Cyrillic fonts
* **DOCX** — editable document generated through Pandoc with an automatic cover page

## Quality-assurance and Security

Checks run automatically in CI.

* **Vale** — AsciiDoc style linter (vale.xml)
* **htmltest** — link-integrity validation (htmltest.log)
* **Shellcheck** — Bash-script analysis
* **Security audit** — OSV-Scanner, Sandworm, and banned-pattern scan (non-blocking)

# Build Pipeline

Adaptadocx can be built with **Make** (default) or an **equivalent Python script**. Both produce identical artefacts in build/.

## Build modes

**Local** (default) — builds only the current branch HEAD or a branch set by BUILD\_REF.

make build-all  
make build-all BUILD\_REF=my-feature

**Tags** — multiversion build over all Git tags.

make build-all BUILD\_SCOPE=tags

Outputs are versioned and stored in:

* build/pdf/<locale>/<version>/adaptadocx-<locale>.pdf
* build/docx/<locale>/<version>/adaptadocx-<locale>.docx
* copied to site/<locale>/<version>/\_downloads/

The header menu links **PDF** and **DOCX** always point to \_downloads of the current version.

## Docker Environment

The container image includes:

* Node.js 20 and Ruby ≥ 2.7
* Python 3.11+
* Asciidoctor PDF
* Vale, htmltest, Shellcheck
* Graphviz
* DejaVu fonts
* rsvg-convert for SVG→PDF/SVG→PNG conversion

## Makefile Targets

| Target | Purpose |
| --- | --- |
| make build-all | Build **HTML + PDF + DOCX** (alias of build-site) |
| make build-html | Build HTML only |
| make build-pdf | Build PDF only |
| make build-docx | Build DOCX only |
| make test | Run Vale, htmltest and Shellcheck |
| make clean | Remove build/ |
| make release | Zip artefacts after QA |

## Python build script

If make is not available, run the same tasks via build.py (Python 3.11+).

| Command | Purpose |
| --- | --- |
| python3 build.py build-site | HTML + PDF + DOCX |
| python3 build.py build-html | HTML only |
| python3 build.py build-pdf | PDF only |
| python3 build.py build-docx | DOCX only |
| python3 build.py test | Run tests (Vale, htmltest if site exists, Shellcheck) |
| python3 build.py prep | Version substitution only |
| python3 build.py clean | Remove build/ |

Use either entry point — mixing them in one run is unnecessary.

## Continuous Integration

GitHub Actions defines three workflow groups.

1. **QA Checks** — lints AsciiDoc, validates links, analyses scripts
2. **Security Audit** — dependency and content audit (OSV-Scanner, Sandworm, banned patterns)
3. **Release** — full multiversion build (BUILD\_SCOPE=tags), packaging, and deployment to Netlify

# Getting Started

## Installation (Docker, recommended)

docker build -t adaptadocx .  
# Makefile path  
docker run --rm -v "$(pwd)":/work adaptadocx make build-all  
# Python path  
docker run --rm -v "$(pwd)":/work adaptadocx python3 build.py build-site

## Local Installation

npm ci --no-audit --no-fund  
# using Make  
make build-all  
# or using Python  
python3 build.py build-site

# Next Steps

* [Quick Start](#quick-start)
* [Installation](#installation)

# Quick Start

Follow these steps to build Adaptadocx in all three formats **HTML**, **PDF**, and **DOCX** and run the QA checks.

## Prerequisites

Perform the [Installation](#installation).

## One-shot build (all formats)

**Docker (recommended)**

# Local branch only (default)  
docker run --rm -v "$(pwd)":/work adaptadocx make build-all  
  
# All tags (multiversion)  
docker run --rm -v "$(pwd)":/work adaptadocx make build-all BUILD\_SCOPE=tags

**Local**

# Local branch only (default)  
make build-all  
  
# All tags (multiversion)  
make build-all BUILD\_SCOPE=tags

This creates versioned artefacts:

* build/site/<locale>/<version>/ — HTML
* build/pdf/<locale>/<version>/adaptadocx-<locale>.pdf — PDF
* build/docx/<locale>/<version>/adaptadocx-<locale>.docx — DOCX
* site/<locale>/<version>/\_downloads/ — published downloads per version

## Verify artefacts

tree -L 3 build/

Expected outline (example):

build/  
├── site/  
│ ├── en/  
│ │ └── 0.1.2/  
│ └── ru/  
│ └── 0.1.2/  
├── pdf/  
│ ├── en/  
│ │ └── 0.1.2/  
│ └── ru/  
│ └── 0.1.2/  
└── docx/  
 ├── en/  
 │ └── 0.1.2/  
 └── ru/  
 └── 0.1.2/

## Build a single format

**HTML only**

make build-html

**PDF only**

make build-pdf  
# Output: build/pdf/<locale>/<version>/adaptadocx-<locale>.pdf  
# Copied to: site/<locale>/<version>/\_downloads/

**DOCX only**

make build-docx  
# Output: build/docx/<locale>/<version>/adaptadocx-<locale>.docx  
# Copied to: site/<locale>/<version>/\_downloads/

## Run quality checks

make test

Tools executed:

* **Vale** → vale.xml
* **htmltest** → htmltest.log
* **Shellcheck** → console output

View results:

cat vale.xml  
cat htmltest.log

## Package for release

make release

This creates adaptadocx-docs-<version>.zip in build/.

## Edit-build-preview loop

1. Edit .adoc files in docs/en/…​ or docs/ru/…​.
2. Clean previous build: make clean
3. Re-build: make build-all
4. Open build/site/en/<version>/index.html (or build/site/en/current/index.html) in a browser; use ru/ for Russian.

## Next steps

* [Architecture](#system-architecture)
* [Installation](#installation)

# Authoring Guide

This guide explains the everyday workflow for writers who create and maintain documentation.

## Overview

* Write content in **AsciiDoc**.
* Validate locally (**Vale** for style, **htmltest** for links, build preview).
* Commit and open a pull request.
* CI produces **HTML** / **PDF** / **DOCX** and runs QA checks.
* Multiversion site per Git tags. Each version exposes its own downloads in \_downloads.

## Project structure

docs/  
├── en/ # English component  
│ ├── antora.yml # Component metadata  
│ └── modules/ROOT/ # Default Antora module  
│ ├── pages/ # AsciiDoc pages  
│ ├── assets/ # Static assets  
│ │ └── images/ # Local images  
│ └── attachments/ # Files for download (go to \_downloads)  
└── ru/ # Russian component  
 ├── antora.yml # Component metadata  
 └── modules/ROOT/  
 ├── pages/ # AsciiDoc pages  
 ├── assets/ # Static assets  
 │ └── images/ # Local images  
 └── attachments/ # Files for download (go to \_downloads)

Each language is an independent Antora component; titles and versions may differ.

## Environment setup

1. Complete [Installation & Setup](#installation).
2. Clone the repository and install Node.js packages:

* npm ci --no-audit --no-fund

1. Perform an initial build to verify the tool-chain:

* make build-all

## Build modes

**Local** — default. Builds only the current branch **HEAD** or the branch set by **BUILD\_REF**.

make build-all  
make build-all BUILD\_REF=my-feature

**Tags** — multiversion build over all Git tags.

make build-all BUILD\_SCOPE=tags

Make variables: **BUILD\_SCOPE** = **local** or **tags**, **BUILD\_REF** defaults to **HEAD**.

## Artefacts and layout

### PDF

* build/pdf/<locale>/<version>/adaptadocx-<locale>.pdf
* Copied to site/<locale>/<version>/\_downloads/adaptadocx-<locale>.pdf

### DOCX

* build/docx/<locale>/<version>/adaptadocx-<locale>.docx
* Copied to site/<locale>/<version>/\_downloads/adaptadocx-<locale>.docx

Header menu links **PDF** and **DOCX** point to \_downloads of the current version.

## Daily workflow

1. **Edit** .adoc files in the appropriate language directory.
2. **Preview** the site:

* make build-html

1. **Validate** style and links:

* make test

1. **Produce all formats** when you need review packages:

* make build-all

1. **Inspect** artefacts in build/ and downloads under site/<locale>/<version>/\_downloads.

## Pre-commit validation

Run the core checks before every commit:

vale docs/  
make build-html  
htmltest -c .htmltest.yml build/site

## Pull-request flow

1. Create a feature branch:

* git checkout -b feat/short-description

1. Make changes and run a local build:

* make build-all && make test

1. Commit, push, and open a PR.

CI attaches vale.xml and htmltest.log to the run.

## CI pipelines

### Release (tags)

* Trigger: push to a tag.
* Build in Docker with BUILD\_SCOPE=tags.
* Validate with **htmltest** and **Vale**.
* Upload built site and deploy to production.

### QA checks (pull requests to main)

* Lint shell scripts, run **Vale**.
* Build and run **htmltest** for the current branch.

### Security audit (pull requests to main, push to tags)

* Non-blocking checks: **OSV-Scanner**, **Sandworm**, banned-pattern scan.
* Summary published in the workflow run.

## Translation workflow

1. Write or update the English page.
2. Copy it to the mirror path under docs/ru/ and translate.
3. Verify cross-references in both languages.
4. Run make build-html and confirm search results appear.
5. Open a pull request.

## Tool summary

| Category | Tools / Key files |
| --- | --- |
| Editing | AsciiDoc-aware editor |
| Validation | **Vale**, **htmltest**, **Shellcheck** |
| Build | Makefile, Dockerfile |
| Config | antora-playbook-en.yml, antora-playbook-ru.yml, antora-assembler.yml, .vale.ini, config/default-theme.yml |
| CI | .github/workflows/release.yml, .github/workflows/qa-checks.yml, .github/workflows/security-audit.yml |

## Related pages

* [Quick Start](#quick-start)
* [Installation](#installation)

# DocOps Guide

## Architecture

Adaptadocx is a modular documentation system built on **Antora**. The stack is organised into five layers:

1. Content sources
2. Build orchestration
3. Format generation (HTML, PDF, DOCX)
4. Quality assurance
5. CI-driven security and packaging

### Core Components

| Layer | Directory / File | Role |
| --- | --- | --- |
| Content | docs/en/, docs/ru/ | Language-specific AsciiDoc sources |
| Content | docs/\*/modules/ROOT/pages | Individual .adoc pages |
| Content | docs/\*/antora.yml | Component name, version, navigation |
| Config | antora-playbook-en.yml, antora-playbook-ru.yml | Site-level config (sources, UI, output) |
| Config | antora-assembler.yml | Assembler settings for PDF/DOCX exports |
| Config | config/ | PDF theme and per-locale metadata (meta-\*.yml) |
| UI | custom-ui/ | CSS/JS and layout overrides (downloads point to \_downloads) |
| Build | Makefile, build.py | Primary / alternative orchestrators |
| Container | Dockerfile | Reproducible tool-chain for CI and local runs |
| CI | .github/workflows/\*.yml | QA, security audit, and release workflows |

### Configuration Precedence

| Priority | File | Scope |
| --- | --- | --- |
| **1** | docs/\*/antora.yml | Component |
| **2** | antora-playbook-\*.yml | Entire site |
| **3** | Inline attributes | Single document |

### Build Pipeline

1. **build-html** — Antora generates the HTML site into build/site/ and assembly exports into build/asm/. Build scope is controlled by **BUILD\_SCOPE** (local by default, or tags) and **BUILD\_REF** (defaults to HEAD).
2. **build-pdf** — consumes build/asm/ and emits versioned PDFs under build/pdf/<locale>/<version>/, then copies them to site/<locale>/<version>/\_downloads/.
3. **build-docx** — consumes build/asm/ and emits versioned DOCXs under build/docx/<locale>/<version>/, then copies them to site/<locale>/<version>/\_downloads/.
4. **make test** — runs Vale, htmltest, Shellcheck.
5. **Packaging** — make release zips the build output (e.g., adaptadocx-docs-<version>.zip).

build/asm/ holds the intermediate assemblies produced by @antora/pdf-extension. build/site/ is the final HTML website.

#### HTML flow

AsciiDoc → Antora → UI bundle → Lunr index → HTML site

* **Search** — Lunr with English/Russian stemmers.
* **SVG** diagrams render natively in the browser.

#### PDF flow

AsciiDoc → Asciidoctor PDF → theme → fonts → PDF

* **Theme** — config/default-theme.yml.
* **Fonts** — DejaVu (Latin + Cyrillic).
* **SVG** — pre-converted with rsvg-convert when available.

#### DOCX flow

AsciiDoc → Pandoc → Lua filter → metadata → DOCX

**Command (EN example)**

pandoc --from asciidoc \  
 --to docx \  
 --lua-filter=docx/coverpage.lua \  
 --metadata-file=config/meta-en.yml \  
 -o build/docx/en/0.1.2/adaptadocx-en.docx \  
 docs/en/modules/ROOT/pages/\*.adoc

### Quality Assurance

| Tool | Checks | Output |
| --- | --- | --- |
| **Vale** | Style, spelling | vale.xml |
| **htmltest** | Link integrity | htmltest.log |
| **Shellcheck** | Shell-script lint | Console |

### Build Scope Controls

| Variable | Meaning | Default |
| --- | --- | --- |
| BUILD\_SCOPE | local — build only the current branch; tags — build all Git tags | local |
| BUILD\_REF | Branch/ref to build in local mode (e.g., my-feature, HEAD) | HEAD |

### Container Image

* **Base** — Node.js 20 + Ruby ≥ 2.7
* **Extras** — Asciidoctor PDF, Pandoc, Vale, htmltest, Shellcheck, Graphviz, DejaVu fonts

The image is built once and reused by all CI jobs.

### CI Workflows

| Workflow | What it does | Trigger |
| --- | --- | --- |
| QA Checks | Vale, htmltest, Shellcheck | pull\_request (to main) |
| Security Audit | OSV-Scanner, Sandworm, banned-pattern scan | pull\_request → main, push → tags ('\*') |
| Release | Docker build, multiversion docs (BUILD\_SCOPE=tags), zip artefacts, deploy | push to tags |

Release uploads a full site build and packaged artefacts.

### Related Pages

* [Build Orchestration](#build-orchestration)
* [Document Generation](#document-generation)
* [CI/CD Workflows](#ci-cd-workflows)

## Installation

Adaptadocx can run either inside a Docker container (recommended) or on a locally configured toolchain.

### System Requirements

#### Docker

* **Docker Engine 20.10+** (or Docker Desktop)
* ~4 GB free disk space
* Internet access to pull the image

#### Local

* **Node.js 20+** with npm
* **Python 3.11+** (optional, for build.py script)
* **Ruby ≥ 2.7** (required by Asciidoctor PDF)
* **Graphviz** — renders SVG and other diagrams in PDF/DOCX
* **Vale** — style and grammar linting
* **htmltest** — link-integrity validation
* **Shellcheck** — shell-script analysis
* **Git**
* ~2 GB free disk space

### Docker Installation (recommended)

1. Clone the repository

* git clone https://github.com/mikhail-marutyan/adaptadocx.git  
  cd adaptadocx

1. Build the image

* docker build -t adaptadocx .

1. Generate the full documentation set

* # Local branch only (default)  
  docker run --rm -v "$(pwd)":/work adaptadocx make build-all  
    
  # All tags (multiversion)  
  docker run --rm -v "$(pwd)":/work adaptadocx make build-all BUILD\_SCOPE=tags

### Local Installation

1. Install Node.js dependencies

* npm ci --no-audit --no-fund

1. Install Ruby + Asciidoctor PDF

* # Debian / Ubuntu  
  sudo apt-get update  
  sudo apt-get install -y ruby ruby-dev  
  gem install asciidoctor-pdf

1. Install QA tools and Graphviz

* # Vale  
  wget -qO- https://github.com/errata-ai/vale/releases/download/v2.29.4/vale\_2.29.4\_Linux\_64-bit.tar.gz \  
  | tar -xz && sudo mv vale /usr/local/bin/  
    
  # htmltest  
  wget -qO- https://github.com/wjdp/htmltest/releases/download/v0.17.0/htmltest\_0.17.0\_linux\_amd64.tar.gz \  
  | tar -xz && sudo mv htmltest /usr/local/bin/  
    
  # Shellcheck  
  sudo apt-get install -y shellcheck  
    
  # Graphviz  
  sudo apt-get install -y graphviz # Linux  
  brew install graphviz # macOS

1. Build all formats

* # Local branch only (default)  
  make build-all  
    
  # All tags (multiversion)  
  make build-all BUILD\_SCOPE=tags

1. Run quality checks

* make test

### Output locations

* **HTML** — build/site/<locale>/<version>/
* **PDF** — build/pdf/<locale>/<version>/adaptadocx-<locale>.pdf
* **DOCX** — build/docx/<locale>/<version>/adaptadocx-<locale>.docx
* Published downloads — site/<locale>/<version>/\_downloads/

### Troubleshooting

#### Docker

* **Build permission error** — confirm the Docker daemon is running and your user is in the **docker** group.
* **Volume write error** — check file-system permissions on the mounted path.

#### Local

* **Incorrect Node.js version** — use **nvm** to install Node.js 20.
* **Ruby gem fails** — ensure ruby-dev and a C compiler are present.
* **Vale / htmltest not found** — verify they are in $PATH.

#### Build

* **PDF font issues** — install DejaVu fonts (sudo apt-get install -y fonts-dejavu).
* **SVG missing in PDF** — install rsvg-convert (sudo apt-get install -y librsvg2-bin).
* **QA failures** — review vale.xml, htmltest.log, and Shellcheck output.

### Next Steps

* [Quick Start](#quick-start)
* [Architecture](#system-architecture)

## Document Generation

Adaptadocx produces three artefact types from the AsciiDoc components:

* **HTML** — static web site with navigation and search.
* **PDF** — print-ready booklet with a custom theme.
* **DOCX** — editable document.

All pipelines share the same content tree and the Antora **assembler** output, then apply format-specific steps. Outputs are versioned per locale and per tag.

### HTML Pipeline

**HTML flow**

AsciiDoc → Antora → UI bundle → Lunr index → HTML site

#### Antora playbook (EN)

File antora-playbook-en.yml

site:  
 title: Adaptadocx Documentation  
 start\_page: en::index.adoc  
 url: https://adaptadocx.netlify.app/en  
  
content:  
 branches: ~  
 tags: '\*'  
 sources:  
 - url: .  
 start\_path: docs/en  
  
urls:  
 html\_extension\_style: default  
  
ui:  
 bundle:  
 url: ./custom-ui/ui-bundle.zip  
 snapshot: true  
 supplemental\_files: ./custom-ui/supplemental\_ui  
  
output:  
 dir: ./build/site  
  
antora:  
 extensions:  
 - require: '@antora/pdf-extension'  
 configFile: ./antora-assembler.yml  
 - require: '@antora/lunr-extension'  
 languages: [en]

* **Search** — Lunr is enabled per locale (see languages).
* **UI bundle** — custom-ui/ overrides layouts, CSS, JS. Header links point to versioned downloads under \_downloads.

#### Assembler configuration

File antora-assembler.yml

assembly:  
 root\_level: 0  
 section\_merge\_strategy: fuse  
 xml\_ids: true  
component\_version\_filter:  
 names: '\*\*'  
build:  
 dir: ./build/asm  
 keep\_source: true  
 command: 'true'  
 publish: false  
 qualify\_exports: true

The assembler produces exported trees under build/asm/<locale>/<version>/\_exports/.

### PDF Pipeline

**PDF flow**

AsciiDoc (from assembler) → Asciidoctor PDF → theme → fonts → PDF

**For each** locale and version discovered in site/<locale>/<version>/:

1. Resolve the exported entry file build/asm/<locale>/<version>/\_exports/index.adoc.
2. Copy images from build/asm/<locale>/<version>/\_images if present.
3. Render with Asciidoctor PDF and version-specific revnumber.

Theme file config/default-theme.yml:

extends: default  
font:  
 catalog:  
 DejaVu Sans:  
 normal: DejaVuSans.ttf  
 bold: DejaVuSans-Bold.ttf  
 italic: DejaVuSans-Oblique.ttf  
 bold\_italic: DejaVuSans-BoldOblique.ttf  
 DejaVu Sans Mono:  
 normal: DejaVuSansMono.ttf  
 bold: DejaVuSansMono-Bold.ttf  
page:  
 size: A4  
 margin: [2cm, 2cm, 2cm, 2cm]  
base:  
 font-family: DejaVu Sans  
 font-size: 11

#### Output layout

* build/pdf/<locale>/<version>/adaptadocx-<locale>.pdf
* copied to site/<locale>/<version>/\_downloads/adaptadocx-<locale>.pdf

### DOCX Pipeline

**DOCX flow**

AsciiDoc (from assembler) → Asciidoctor DocBook → Pandoc → Lua filters → DOCX

**For each** locale and version:

1. Read build/asm/<locale>/<version>/\_exports/index.adoc.
2. Convert to DocBook via Asciidoctor.
3. Pipe into Pandoc with reference DOCX and metadata for the target locale/version.
4. Optionally convert SVGs to PNG if rsvg-convert is available.

Example (conceptually mirrors the Makefile):

# Inside CI the Makefile loops locales/versions and runs something equivalent to:  
(cd "build/asm/<locale>/<version>/\_exports" && \  
 asciidoctor -b docbook5 \  
 -r extensions/collapsible\_tree\_processor.rb \  
 -a allow-uri-read -a revdate! -a revnumber! -a docdate! -a docdatetime! \  
 -o - index.adoc \  
| pandoc --from=docbook --to=docx \  
 --reference-doc=docx/reference.docx \  
 --metadata-file=config/meta-<locale>.yml \  
 --lua-filter=docx/coverpage.lua \  
 $( [ -x "$(command -v rsvg-convert)" ] && echo "--lua-filter=docx/svg2png.lua" ) \  
 -o "build/docx/<locale>/<version>/adaptadocx-<locale>.docx")

**Output layout**: \* build/docx/<locale>/<version>/adaptadocx-<locale>.docx \* copied to site/<locale>/<version>/\_downloads/adaptadocx-<locale>.docx

#### Cover-page filter

File docx/coverpage.lua

function Meta(meta)  
 meta.version = meta.version or os.getenv('VERSION') or 'dev'  
 return meta  
end

### Versioning rules

* Local builds default to the current branch HEAD (or BUILD\_REF) — single version per locale.
* Release builds use all Git tags (BUILD\_SCOPE=tags) — multiple versions per locale.
* UI header links resolve to \_downloads/adaptadocx-<locale>.(pdf|docx) inside the current version folder.

### Troubleshooting

* **HTML** — broken links → run make test and inspect htmltest.log.
* **PDF** — missing export file → ensure build/asm/<locale>/<version>/\_exports/index.adoc exists for that version.
* **DOCX** — Pandoc parse errors → verify Lua filters (docx/coverpage.lua, docx/svg2png.lua) and the DocBook stream.
* **Missing fonts** → install fonts-dejavu in your environment.

### Related Pages

* [Architecture](#system-architecture)
* [CI/CD Workflows](#ci-cd-workflows)
* [Build Orchestration](#build-orchestration)

## Build Orchestration

### Overview

The build layer is deliberately **duplicated**:

* **GNU Make** — primary entry-point used by CI and Docker
* **build.py** — Python 3.11+ equivalent for systems without **make**

Both entry-points drive the same pipeline:

* **Generation** — HTML / PDF / DOCX
* **Quality assurance** — Vale, htmltest, Shellcheck
* **Packaging** — ZIP archive per build
* **Containerisation** — fully reproducible via docker build

Use either tool — **never mix them in one run**.

### Build modes

**Local** — builds only the current branch HEAD or a branch set via BUILD\_REF.

make build-all  
make build-all BUILD\_REF=my-feature

**Tags** — multiversion build over all Git tags.

make build-all BUILD\_SCOPE=tags

Make variables: BUILD\_SCOPE = local or tags; BUILD\_REF defaults to HEAD.

### Artefacts and layout

#### PDF

* build/pdf/<locale>/<version>/adaptadocx-<locale>.pdf
* copied to site/<locale>/<version>/\_downloads/adaptadocx-<locale>.pdf

#### DOCX

* build/docx/<locale>/<version>/adaptadocx-<locale>.docx
* copied to site/<locale>/<version>/\_downloads/adaptadocx-<locale>.docx

### Make targets

| Target | Purpose |
| --- | --- |
| make build-site | HTML + PDF + DOCX (all locales) |
| make build-html | HTML only |
| make build-pdf | PDF only |
| make build-docx | DOCX only |
| make test | Vale • htmltest • Shellcheck |
| make clean | Remove build/ |
| make release | ZIP artefacts + QA |
| make build-all | Alias → **build-site** |

### Python entry-points

| Command | Purpose |
| --- | --- |
| python3 build.py build-site | HTML + PDF + DOCX |
| python3 build.py build-html | HTML only |
| python3 build.py build-pdf | PDF only |
| python3 build.py build-docx | DOCX only |
| python3 build.py test | Run tests (Vale, htmltest if site exists, Shellcheck) |
| python3 build.py clean | Remove build/ |

**Options and defaults**:

* --scope local|tags (default local) **local** builds only the current ref; **tags** builds all Git tags.
* --ref <git-ref> (default HEAD) Used only when --scope local.
* Environment variables mirror the flags: BUILD\_SCOPE, BUILD\_REF.

**Examples**:

# All tags (multiversion)  
python3 build.py --scope tags build-all  
  
# Local build for a specific branch  
python3 build.py --scope local --ref my-feature build-site

### Primary Make targets (abridged)

# Full build  
build-site: build-html build-pdf build-docx  
 @echo "[site] full build done"  
  
# HTML (Antora for all locales; BUILD\_SCOPE/BUILD\_REF control sources)  
build-html:  
 @echo "[html] start"; \  
 for l in $(LOCALES); do \  
 echo " • $${l}"; \  
 pb="antora-playbook-$${l}.yml"; \  
 if [ "$(BUILD\_SCOPE)" = "tags" ]; then \  
 npx antora "$$pb"; \  
 else \  
 bak="$$pb.bak"; \  
 cp "$$pb" "$$bak"; \  
 tr -d '\r' < "$$pb" > "$$pb.unix" && mv "$$pb.unix" "$$pb"; \  
 sed -i "s/tags: '\\*'/tags: ~/" "$$pb"; \  
 sed -i "s/branches: ~$$/branches: $(BUILD\_REF)/" "$$pb"; \  
 npx antora "$$pb"; \  
 mv "$$bak" "$$pb"; \  
 fi; \  
 done  
 @echo "[html] done"  
  
# PDF (versioned outputs per locale/version)  
build-pdf: build-html  
 @mkdir -p "$(PDF\_DIR)"; \  
 for l in $(LOCALES); do \  
 echo "[pdf] $$l"; \  
 for version\_dir in $(SITE\_DIR)/$$l/\*/; do \  
 if [ -d "$$version\_dir" ]; then \  
 version=$$(basename "$$version\_dir"); \  
 if [ "$(BUILD\_SCOPE)" != "tags" ] && [ "$$version" != "$(BUILD\_REF)" ] && [ "$$version" != "current" ] && [ "$$version" != "main" ]; then continue; fi; \  
 export\_file=""; \  
 for candidate in "$(ASM\_DIR)/$$l/$$version/\_exports/index.adoc" "$(ASM\_DIR)/$$l/\_exports/index.adoc" "$(ASM\_DIR)/\_exports/$$l/$$version/index.adoc" "$(ASM\_DIR)/\_exports/$$l/index.adoc"; do \  
 if [ -f "$$candidate" ]; then export\_file="$$candidate"; base=$$(dirname "$$(dirname "$$candidate")"); break; fi; \  
 done; \  
 [ -z "$$export\_file" ] && continue; \  
 img\_src="$$base/\_images"; img\_dst="$$(dirname "$$export\_file")/$$l/$$version/\_images"; \  
 [ -d "$$img\_src" ] && mkdir -p "$$img\_dst" && cp -r "$$img\_src"/\* "$$img\_dst"/ || true; \  
 outdir="$(PDF\_DIR)/$$l/$$version"; outfile="$$outdir/adaptadocx-$$l.pdf"; \  
 mkdir -p "$$outdir"; \  
 toc=$$( [ "$$l" = ru ] && echo '-a toc-title=Содержание' || echo '-a toc-title=Contents' ); \  
 asciidoctor-pdf $(ASCIIDOCTOR\_PDF\_OPTS) $$toc -a revnumber=$$version -o "$$outfile" "$$export\_file"; \  
 mkdir -p "$(SITE\_DIR)/$$l/$$version/\_downloads"; \  
 cp "$$outfile" "$(SITE\_DIR)/$$l/$$version/\_downloads/adaptadocx-$$l.pdf"; \  
 fi; \  
 done; \  
 done  
 @echo "[pdf] done"  
  
# DOCX (versioned outputs per locale/version)  
build-docx: build-html  
 @mkdir -p "$(DOCX\_DIR)"; \  
 for l in $(LOCALES); do \  
 echo "[docx] $$l"; \  
 for version\_dir in $(SITE\_DIR)/$$l/\*/; do \  
 if [ -d "$$version\_dir" ]; then \  
 version=$$(basename "$$version\_dir"); \  
 if [ "$(BUILD\_SCOPE)" != "tags" ] && [ "$$version" != "$(BUILD\_REF)" ] && [ "$$version" != "current" ] && [ "$$version" != "main" ]; then continue; fi; \  
 base="$(ASM\_DIR)/$$l/$$version"; \  
 img\_src="$$base/\_images"; img\_dst="$$base/\_exports/$$l/$$version/\_images"; \  
 [ -d "$$img\_src" ] && mkdir -p "$$img\_dst" && cp -r "$$img\_src"/\* "$$img\_dst"/ || true; \  
 outdir="$(DOCX\_DIR)/$$l/$$version"; outfile="$$outdir/adaptadocx-$$l.docx"; outfile\_abs="$(CURDIR)/$$outfile"; \  
 mkdir -p "$$outdir"; \  
 tmp\_meta="$(CURDIR)/$(DOCX\_DIR)/meta-$$l-$$version.yml"; \  
 sed "s/{page-version}/$$version/g" $(CURDIR)/config/meta-$$l.yml > "$$tmp\_meta"; \  
 ( cd "$$base/\_exports" && asciidoctor -b docbook5 -r $(CURDIR)/extensions/collapsible\_tree\_processor.rb -a allow-uri-read -a revdate! -a revnumber! -a docdate! -a docdatetime! -o - index.adoc | pandoc --from=docbook --to=docx --reference-doc=$(PANDOC\_REF) --metadata-file="$$tmp\_meta" $(SVG\_FILTER) --lua-filter=$(LUA\_COVER) -o "$$outfile\_abs" ); \  
 rm -f "$$tmp\_meta"; \  
 mkdir -p "$(SITE\_DIR)/$$l/$$version/\_downloads"; \  
 cp "$$outfile" "$(SITE\_DIR)/$$l/$$version/\_downloads/adaptadocx-$$l.docx"; \  
 fi; \  
 done; \  
 done  
 @echo "[docx] done"

#### QA helpers

test:  
 @if [ -d "$(SITE\_DIR)" ]; then \  
 htmltest -c .htmltest.yml "$(SITE\_DIR)"; \  
 else \  
 echo "[test] Skipping htmltest - no site built"; \  
 fi  
 @vale --config=.vale.ini docs/  
 @find scripts -name '\*.sh' -print0 | xargs -0 -I{} bash -c 'tr -d "\r" < "{}" | shellcheck -'  
 @echo '[test] OK'

#### Service targets

clean:  
 -rm -rf build  
 @echo '[clean] build/ removed'  
  
release: build-site test  
 @cd build && zip -rq ../"$(RELEASE\_FILE)" .  
 @echo "[release] $(RELEASE\_FILE) created"

Where RELEASE\_FILE := adaptadocx-docs-$(VERSION).zip.

### Docker workflow

Docker image encapsulates the tool-chain; typical runs:

# Build image  
docker build -t adaptadocx:latest .  
  
# Full build  
docker run --rm -v "$(pwd)":/work adaptadocx:latest make build-site  
  
# QA-only  
docker run --rm -v "$(pwd)":/work adaptadocx:latest make test  
  
# Interactive debugging  
docker run -it --rm -v "$(pwd)":/work adaptadocx:latest bash

### Configuration variables

|  |  |  |
| --- | --- | --- |
| Variable | Role | Default |
| LOCALES | Supported languages | ru en |
| VERSION | Version from Git/package.json | auto-detected |
| BUILD\_SCOPE | Build mode (local or tags) | local |
| BUILD\_REF | Branch to build in local mode | HEAD |
| SITE\_DIR | HTML site directory | build/site |
| ASM\_DIR | Antora assembly directory | build/asm |
| PDF\_DIR | PDF output directory | build/pdf |
| DOCX\_DIR | DOCX output directory | build/docx |
| PANDOC\_REF | Reference DOCX | docx/reference.docx |
| LUA\_COVER | Cover page Lua filter | docx/coverpage.lua |
| SVG\_FILTER | SVG→PNG Lua filter | docx/svg2png.lua (if available) |
| RELEASE\_FILE | Release archive name | adaptadocx-docs-$(VERSION).zip |

#### Version detection

VERSION := $(shell git describe --tags --abbrev=0 2>/dev/null \  
 || node -p "require('./package.json').version")

### Troubleshooting

* **Unknown target** — run make from repo root
* **Stale artefacts** — run make clean before next build
* **CI drift** — ensure Docker tool versions match local ones

See also: [CI/CD Workflows](#ci-cd-workflows)

## Title Management

Titles are resolved in a strict three-level hierarchy:

| Priority | Source | Scope | Override |
| --- | --- | --- | --- |
| **1** | Component file docs/\*/antora.yml | Component name / version | Overrides all |
| **2** | Playbook antora-playbook-\*.yml | Site branding | Overrides documents |
| **3** | Attributes inside .adoc | Single file | Lowest |

### Component-level titles

File docs/en/antora.yml

name: en  
title: Adaptadocx Documentation  
version: '1.0'  
display\_version: '1.0'  
  
asciidoc:  
 attributes:  
 component-title: '{title}'  
 component-version: '{version}'  
 document-title: '{component-title} {component-version}'

The Russian component docs/ru/antora.yml mirrors this, using localized strings and name: ru.

| Attribute | Purpose |
| --- | --- |
| title | Primary label in the left nav |
| version | Machine-readable version |
| display\_version | Human-readable version |
| component-title | Reusable attribute |
| document-title | Template for derived titles |

### Playbook-level titles

Excerpt antora-playbook-en.yml

site:  
 title: Adaptadocx Documentation  
 start\_page: en::index.adoc  
  
asciidoc:  
 attributes:  
 site-title: '{site.title}'  
 page-title-pattern: '{site-title} - {page-title}'

The Russian playbook antora-playbook-ru.yml uses localized values.

| Attribute | Purpose |
| --- | --- |
| site.title | Main HTML <title> |
| site-title | Reusable attribute |
| page-title-pattern | Global HTML title template |

### Document-level titles

Inside any .adoc file:

= System Architecture  
:navtitle: Architecture

| Attribute | Effect | May be overridden |
| --- | --- | --- |
| = Title | H1 in rendered content | Metadata can change |
| :navtitle: | Menu entry | Component nav may replace |

### Output handling

* **HTML** — <title> is resolved via page-title-pattern → component → document.
* **PDF** — title block uses component values; version label comes from asciidoctor-pdf with -a revnumber=<version> (Makefile passes the site version).
* **DOCX** — metadata is loaded from per-locale files config/meta-en.yml / config/meta-ru.yml; the Makefile generates per-version temporary files build/docx/meta-<locale>-<version>.yml by replacing 0.1.0.

### Troubleshooting

* **Inconsistent titles** — ensure component title/version and playbook attributes are aligned.
* **Wrong PDF title/version** — check -a revnumber usage in the Makefile and config/default-theme.yml.
* **Nav vs. page mismatch** — adjust :navtitle: or the component nav.adoc.

### Related Pages

* [Architecture](#system-architecture)
* [CI/CD Workflows](#ci-cd-workflows)
* [Build Orchestration](#build-orchestration)

## Version Management

Adaptadocx coordinates releases across languages using **semantic versioning**, Antora component metadata, and Git-tagged CI workflows.

### Semantic versioning

Version string: **MAJOR.MINOR.PATCH**

| Segment | Meaning |
| --- | --- |
| MAJOR | Breaking changes |
| MINOR | Backward-compatible features |
| PATCH | Bug fixes / docs only |

**Example timeline**

1.0.0 → initial release  
1.1.0 → new features  
1.1.1 → fixes / docs  
2.0.0 → breaking refactor

### Git guidelines

**Tag and push**

git tag -a v1.2.0 -m "Release 1.2.0"  
git push origin --tags

Branch types

| Branch | Purpose | Pattern |
| --- | --- | --- |
| main | Stable code | main |
| Feature | New work | feature/\* |
| Release | Preparation branch (rare) | release/vX.Y.Z |
| Hot-fix | Urgent patch | hotfix/vX.Y.Z |

### Antora component versions

Each language keeps its own docs/en/antora.yml and docs/ru/antora.yml. For **tag builds** Antora resolves the version from Git tags; for **branch builds** the content is published under a current (or branch-named) pseudo-version. No placeholders are required in the repo.

**English**

name: adaptadocx  
version: 'current' # tag builds override this automatically  
title: Adaptadocx Documentation

**Russian**

name: adaptadocx  
version: 'current' # tag builds override this automatically  
title: Документация Adaptadocx

### Version resolution logic

The build uses a fallback so both tagged releases and branch builds resolve correctly.

**Directory layout**

build/asm/  
├── <locale>/  
│ ├── 1.2.0/ ← Git-tagged release v1.2.0  
│ └── current/ ← Untagged builds from the branch

1. Look for …/<version>/.
2. If not found, use …/current/.

### CI and build scope

* **Local/PR builds** — default BUILD\_SCOPE=local, only the current branch (BUILD\_REF=HEAD).
* **Release builds** — BUILD\_SCOPE=tags, all Git tags are built by Antora.

Example (release workflow):

- name: Build docs in container  
 run: |  
 docker run --rm -v "${{ github.workspace }}:/work" adaptadocx:latest \  
 bash -lc 'npm ci --no-audit --prefer-offline && make clean && make build-all BUILD\_SCOPE=tags'

### Artefact layout

Versioning is encoded in **directories**, filenames stay stable.

| Format | Output path |
| --- | --- |
| PDF (EN) | build/pdf/en/<version>/adaptadocx-en.pdf → copied to site/en/<version>/\_downloads/adaptadocx-en.pdf |
| PDF (RU) | build/pdf/ru/<version>/adaptadocx-ru.pdf → copied to site/ru/<version>/\_downloads/adaptadocx-ru.pdf |
| DOCX (EN) | build/docx/en/<version>/adaptadocx-en.docx → copied to site/en/<version>/\_downloads/adaptadocx-en.docx |
| DOCX (RU) | build/docx/ru/<version>/adaptadocx-ru.docx → copied to site/ru/<version>/\_downloads/adaptadocx-ru.docx |

Header menu links in the UI point at \_downloads/ of the **current version**, for example:

* '\_downloads/adaptadocx-en.pdf'
* '\_downloads/adaptadocx-en.docx'
* '\_downloads/adaptadocx-ru.pdf'
* '\_downloads/adaptadocx-ru.docx'

### Translation release checklist

1. Bump the app version in package.json if applicable.
2. Verify playbooks if branches or start\_path changed.
3. Build both locales and verify search indexing.
4. Tag and push (vX.Y.Z).
5. Confirm CI artefacts exist for **EN** and **RU** under site/<locale>/<version>/\_downloads/ and build/(pdf|docx)/<locale>/<version>/.

### Related pages

* [Architecture](#system-architecture)
* [CI/CD Workflows](#ci-cd-workflows)
* [Build Orchestration](#build-orchestration)

## CI/CD Workflows

Adaptadocx automates linting, QA, security checks, and packaged builds with **GitHub Actions**. Artefacts are delivered as a ZIP and as versioned downloads inside the site.

### Workflow Matrix

| Workflow | Trigger | Jobs |
| --- | --- | --- |
| QA Checks | pull\_request → **main** | **Shellcheck** · **Vale** · **htmltest** (parallel), build in Docker |
| Security Audit | pull\_request → **main**, push → tags ('\*') | **OSV-Scanner** · **Sandworm** · banned-pattern scan (non-blocking) |
| Release | push → tags ('\*') | Docker build → make build-all BUILD\_SCOPE=tags → **htmltest** + **Vale** → ZIP + upload artefacts |
| Deploy | after **Release** on tag | Download built site → Netlify deploy --prod |

### QA Checks

File: /.github/workflows/qa-checks.yml

* Jobs: **shellcheck**, **vale**, **htmltest**.
* Trigger: pull\_request to **main**.
* Each job runs with timeouts and uploads reports on failure.
* HTML for **htmltest** is built inside the same Docker image as in release.

Build step in the HTML testing job:

- name: Build docs image  
 run: docker build -t adaptadocx:latest .  
  
- name: Build docs in container  
 run: |  
 docker run --rm \  
 -v "${{ github.workspace }}:/work" \  
 adaptadocx:latest \  
 bash -lc 'npm ci --no-audit --prefer-offline && make clean && make build-all'

Reports are always uploaded (examples):

- name: Upload htmltest log  
 if: always()  
 uses: actions/upload-artifact@v4  
 with:  
 name: htmltest-log  
 path: htmltest.log

### Security Audit

File: /.github/workflows/security-audit.yml

* Trigger: pull\_request → **main** and push to tags ('\*').
* Steps: **OSV-Scanner**, **Sandworm audit**, **banned-pattern** scan, then a short summary to $GITHUB\_STEP\_SUMMARY.
* Behavior: all checks use continue-on-error: true — the audit warns but does not block the PR.
* Outputs:
  + reports/osv.json — results from OSV (skipped if no lockfiles are present)
  + reports/sandworm.json — @sandworm/audit report
  + reports/banned-patterns-report.txt — custom grep-gate results

Key snippets:

**OSV-Scanner**

- name: OSV scan  
 id: osv  
 continue-on-error: true  
 run: |  
 files=$(git ls-files | grep -E 'package-lock\.json$|pnpm-lock\.yaml$|yarn\.lock$' || true)  
 if [[ -z "$files" ]]; then  
 echo "scanned=false" >> "$GITHUB\_OUTPUT"  
 exit 0  
 fi  
 docker run --rm -v "$PWD:/src" -w /src ghcr.io/google/osv-scanner:latest \  
 --format json --output /src/reports/osv.json $files || true  
 echo "scanned=true" >> "$GITHUB\_OUTPUT"

**Sandworm audit**

- name: Sandworm audit  
 id: sandworm  
 continue-on-error: true  
 run: npx -y @sandworm/audit@latest --json > reports/sandworm.json

**Banned patterns**

- name: Banned patterns  
 id: banned  
 continue-on-error: true  
 run: node scripts/scan-banned-patterns.cjs

**Summary**

- name: Summarise results  
 if: always()  
 run: |  
 echo '### Security audit summary' >> "$GITHUB\_STEP\_SUMMARY"  
 hits=$(grep -c '^BANNED' reports/banned-patterns-report.txt 2>/dev/null || echo 0)  
 echo "\*\*Banned-pattern hits:\*\* $hits" >> "$GITHUB\_STEP\_SUMMARY"  
 if [[ "${{ steps.osv.outputs.scanned }}" == "true" ]]; then  
 echo 'OSV scan ✔' >> "$GITHUB\_STEP\_SUMMARY"  
 else  
 echo 'OSV scan ⏭ (skipped)' >> "$GITHUB\_STEP\_SUMMARY"  
 fi  
 [[ -f reports/sandworm.json ]] \  
 && echo 'Sandworm scan ✔' >> "$GITHUB\_STEP\_SUMMARY" \  
 || echo 'Sandworm scan ✖' >> "$GITHUB\_STEP\_SUMMARY"

### Release

File: /.github/workflows/release.yml

Two jobs: **build** and **deploy**.

#### Build

* Builds the Docker image.
* Runs a full **multiversion** build over tags via BUILD\_SCOPE=tags.
* Validates with **htmltest** and **Vale** in-container.
* Uploads logs and the built site.
* Packs build/ into docs-${{ github.sha }}.zip.

Snippet:

- name: Build docs image  
 run: docker build -t adaptadocx:latest .  
  
- name: Build docs in container  
 run: |  
 docker run --rm \  
 -v "${{ github.workspace }}:/work" \  
 adaptadocx:latest \  
 bash -lc 'npm ci --no-audit --prefer-offline && make clean && make build-all BUILD\_SCOPE=tags'

#### Deploy

Triggered only for tag pushes. Publishes the previously uploaded site to Netlify.

deploy:  
 needs: build  
 runs-on: ubuntu-latest  
 if: github.event\_name == 'push' && github.ref && startsWith(github.ref, 'refs/tags/')  
 steps:  
 - name: Download built site  
 uses: actions/download-artifact@v4  
 with:  
 name: built-site  
 path: site  
  
 - name: Deploy to Netlify  
 run: |  
 npx netlify-cli deploy \  
 --dir=site \  
 --site="${{ secrets.NETLIFY\_SITE\_ID }}" \  
 --auth="${{ secrets.NETLIFY\_AUTH\_TOKEN }}" \  
 --prod

### What gets built

* Local QA builds the current branch (default Make mode BUILD\_SCOPE=local) and runs **htmltest** on build/site.
* Release builds **all tags** (BUILD\_SCOPE=tags) so that versioned downloads are available per tag under:
  + site/<locale>/<version>/\_downloads/
  + corresponding artefacts under build/pdf/<locale>/<version>/ and build/docx/<locale>/<version>/.

### Debugging tips

* Reproduce a failing step locally:
* docker build -t adaptadocx:latest .  
  docker run -it --rm -v "$PWD":/work adaptadocx:latest bash
* Inspect Make execution graph: make -d build-all
* Ensure the runner has full Git history and tags (actions/checkout@v4 with fetch-depth: 0 and git fetch --tags origin).

# FAQ

This FAQ collects common questions about installing, configuring, using, and troubleshooting Adaptadocx.

## Installation and Setup

**What are the minimum system requirements for Adaptadocx?**

Adaptadocx requires:

* **Node.js 18+** with the npm package manager.
* **Ruby ≥ 2.7** — used by Asciidoctor PDF.
* **4 GB free disk space** — repository + build artefacts.
* **Git** — fetches the repository and provides version control.

A Docker-based setup reduces host prerequisites to **Docker Engine 20.10+** and 4 GB of free disk space.

**Should I use Docker or a local installation?**

Docker is recommended in most cases because it provides:

* **Consistency** — identical toolchain and versions on every platform.
* **Simplicity** — the container bundles all dependencies.
* **Isolation** — no interference with existing tools on the host.
* **Reliability** — the image is tested and known to build successfully.

Choose a local installation only when you explicitly need the tools on your host or must integrate with an existing environment.

**How do I verify that my installation works?**

Run a minimal HTML build and check the output.

# Docker workflow  
docker run --rm -v "$(pwd)":/work adaptadocx:latest make build-html  
  
# Local workflow  
make build-html

A successful build places HTML artefacts in:

* build/site/en/<version>/
* build/site/ru/<version>/

You may also see a current/ symlink-like version depending on your Antora config.

## Build System

**What causes "No rule to make target" errors?**

The Makefile could not find the requested rule. Common causes:

* **Wrong directory** — run make from the repository root.
* **Missing Makefile** — confirm that Makefile exists and is readable.
* **Typographical error** — list available targets with:
* grep -E '^[A-Za-z0-9\_-]+:' Makefile | cut -d: -f1 | sort -u
* **File-system permissions** — ensure that Makefile is readable by the current user.

## Output Formats

**Why do Cyrillic characters look incorrect in PDF?**

Ensure that DejaVu fonts are available to the host running Asciidoctor PDF.

# Debian / Ubuntu  
sudo apt-get install -y fonts-dejavu fonts-dejavu-extra  
  
# Verify presence  
fc-list | grep -i dejavu  
  
# Confirm theme references  
grep -i dejavu config/default-theme.yml

**How can I customise PDF styling?**

Edit the theme file config/default-theme.yml. All Asciidoctor PDF theme keys are supported (font families, heading sizes, margins, etc.).

**Where are the generated PDFs and DOCXs saved?**

Each build produces **versioned** outputs per locale:

* **PDF** — build/pdf/<locale>/<version>/adaptadocx-<locale>.pdf
* **DOCX** — build/docx/<locale>/<version>/adaptadocx-<locale>.docx

They are also copied to the site downloads:

* site/<locale>/<version>/\_downloads/adaptadocx-<locale>.pdf
* site/<locale>/<version>/\_downloads/adaptadocx-<locale>.docx

**Can I customise DOCX cover pages?**

Yes — adjust the Lua filter docx/coverpage.lua. For example, to ensure a title and version are always present:

function Meta(meta)  
 meta.title = meta.title or pandoc.MetaString(os.getenv('DOC\_TITLE') or 'Adaptadocx Documentation')  
 meta.version = meta.version or os.getenv('VERSION') or 'dev'  
 return meta  
end

**How do I fix broken links in the HTML output?**

Run link validation and correct failing references.

# Build HTML then test links  
make build-html  
htmltest -c .htmltest.yml build/site  
  
# Review the log  
cat htmltest.log

## Multilingual Documentation

**Can I add languages other than English and Russian?**

Yes — follow the established component pattern:

1. Create a new directory docs/<locale>/.
2. Add docs/<locale>/antora.yml for the component.
3. Add a playbook antora-playbook-<locale>.yml.
4. Provide locale metadata in config/meta-<locale>.yml.
5. Configure search stemming for the new language in the Lunr extension.

## CI/CD and Deployment

**Why do GitHub Actions builds fail while local builds succeed?**

Typical causes include:

* **Environment differences** — CI may have newer or older tool versions.
* **Case sensitivity** — Windows file names vs. case-sensitive Linux runners.
* **Resource limits** — memory constraints or job timeouts.
* **Missing secrets** — environment variables not configured in repository settings.

Enable verbose logging (set -x, --debug, or similar flags) in workflow steps to pinpoint the issue.

## Migration and Upgrades

**How do I migrate existing documentation to Adaptadocx?**

Use the following phased approach:

1. **Content conversion** — translate existing material to AsciiDoc.
2. **Structure organisation** — arrange files into Antora components and modules.
3. **Configuration** — create component descriptors and playbooks.
4. **Link updates** — replace hard-coded paths with xref syntax.
5. **Testing** — run make build-all && make test to validate the result.

## Additional Resources

* [Architecture](#system-architecture)
* [CI/CD Workflows](#ci-cd-workflows)
* [Build Orchestration](#build-orchestration)