Adac Library — Usage Guide

Adac is the core .NET library for working with **ADAC (Archival Digital Asset Container)** files. It provides services for creating, reading, validating, and verifying `.adac` containers — self-describing, ZIP-based archival packages designed for long-term digital preservation.

The library is the foundation used by both the CLI and the REST API.

Table of Contents

- Prerequisites
- Installation
- Quick Start
- Service Registration
- Services
  - IAdacContainerWriter — Creating Containers
  - IAdacContainerReader — Reading and Extracting
  - IAdacValidator — Validating Containers
  - IAdacFixityService — Fixity and Checksums
- Models
  - AdacManifest
  - AdacCoreMetadata
  - AdacPackageDefinition
  - AdacChecksumManifest and AdacFixityReport
  - AdacProvenanceLog
  - AdacRegionAnnotation
  - AdacEditPipeline
  - AdacEncryptionDescriptor
- Validation

    - Validation Results
    - Validation Options
- Constants
- Typical Workflows
- Project Structure
- License

↑ Back to Top

Prerequisites

- .NET 10 SDK or later.

↑ Back to Top

Installation

Add a project reference from your application:

```
dotnet add reference path/to/Adac/Adac.csproj
```

Or, if the library is published as a NuGet package:

```
dotnet add package Adac
```

↑ Back to Top

Quick Start

```
using Adac.Extensions;
using Adac.Models;
using Adac.Services;
using Adac.Validation;
using Microsoft.Extensions.DependencyInjection;

// 1. Set up DI
ServiceCollection services = new();
services.AddAdacCore();
using ServiceProvider sp = services.BuildServiceProvider();

// 2. Create a container
IAdacContainerWriter writer = sp.GetRequiredService<IAdacContainerWriter>();

AdacPackageDefinition package = new()
{
    Description = "Scanned photograph",
    CreatedBy = "my-app/1.0",
    CoreMetadata = new AdacCoreMetadata
    {
        Title = "Family portrait, 1923",
        Format = "TIFF",
    },
    Masters =
    {
        new AdacMasterSource
        {
            Id = "master_0001",
            SourceFilePath = @"C:\scans\photo.tif",
        },
    },
};

await writer.CreateAsync( @"C:\archives\photo.adac", package );

// 3. Read it back
IAdacContainerReader reader = sp.GetRequiredService<IAdacContainerReader>();

AdacManifest manifest = await reader.ReadManifestAsync( @"C:\archives\photo.adac" );
AdacCoreMetadata metadata = await reader.ReadCoreMetadataAsync( @"C:\archives\photo.ad
```

```
Console.WriteLine( $"Title: {metadata.Title}" );
Console.WriteLine( $"Masters: {manifest.Masters.Count}" );

// 4. Validate
IAdacValidator validator = sp.GetRequiredService<IAdacValidator>();
AdacValidationResult result = await validator.ValidateAsync( @"C:\archives\photo.adac"

Console.WriteLine( result.IsValid ? "VALID" : "INVALID" );

// 5. Verify fixity
IAdacFixityService fixity = sp.GetRequiredService<IAdacFixityService>();
AdacFixityReport report = await fixity.VerifyAsync( @"C:\archives\photo.adac" );

Console.WriteLine( report.IsValid ? "Checksums OK" : "Checksum mismatch detected" );
```

↑ Back to Top

Service Registration

All ADAC services are registered with a single extension method call:

```
using Adac.Extensions;

services.AddAdacCore();
```

This registers the following services as **singletons** (all implementations are stateless and thread-safe):

Interface	Implementation	Purpose
`IAdacContainerWriter`	`AdacContainerWriter`	Creates new `.adac` containers.
`IAdacContainerReader`	`AdacContainerReader`	Reads metadata and extracts files from containers.
`IAdacValidator`	`AdacValidator`	Validates containers against the ADAC 1.0 specification.
`IAdacFixityService`	`AdacFixityService`	Computes and verifies SHA-256 checksums.

The method returns the `IServiceCollection` for chaining:

```
builder.Services
    .AddAdacCore()
    .AddLogging();
```

↑ Back to Top

Services

IAdacContainerWriter — Creating Containers

Creates a new `.adac` container from an `AdacPackageDefinition`. The writer generates the manifest, core metadata, checksum manifest, and the full directory structure automatically.

```
IAdacContainerWriter writer = sp.GetRequiredService<IAdacContainerWriter>();

AdacPackageDefinition package = new()
{
    Description = "Two-page letter",
    CreatedBy = "my-app/1.0",
    CoreMetadata = new AdacCoreMetadata
    {
        Title = "Wartime correspondence, 1945",
        Format = "TIFF",
        Tags = ["correspondence", "wartime"],
        Rights = new AdacRights
        {
            Statement = "Public domain",
            License = "CC0-1.0",
        },
    },
    Masters =
    {
        new AdacMasterSource
        {
            Id = "master_0001",
            SourceFilePath = @"C:\scans\page_001.tif",
        },
        new AdacMasterSource
        {
            Id = "master_0002",
            SourceFilePath = @"C:\scans\page_002.tif",
        },
    },
};

await writer.CreateAsync( @"C:\archives\letter.adac", package );
```

Tracking progress

Pass an `IProgress<AdacOperationProgress>` to receive phase-by-phase updates:

```
Progress<AdacOperationProgress> progress = new( p ⇒
{
    Console.WriteLine( $"[{p.Phase}] ({p.Current}/{p.Total}) {p.CurrentFile}" );
} );

await writer.CreateAsync( @"C:\archives\letter.adac", package, progress );
```

Output:

```
[CopyingMasters] (1/2) page_001.tif
[CopyingMasters] (2/2) page_002.tif
[WritingChecksums] (1/1) provenance/checksums.json
```

Cancellation

All async methods accept an optional `CancellationToken`:

```
using CancellationTokenSource cts = new( TimeSpan.FromSeconds( 30 ) );
await writer.CreateAsync( outputPath, package, cancellationToken: cts.Token );
```

IAdacContainerReader — Reading and Extracting

Reads metadata, lists entries, and extracts files from existing containers.

Reading metadata

```
IAdacContainerReader reader = sp.GetRequiredService<IAdacContainerReader>();

// Manifest (required)
AdacManifest manifest = await reader.ReadManifestAsync( "archive.adac" );

// Core metadata (required)
AdacCoreMetadata metadata = await reader.ReadCoreMetadataAsync( "archive.adac" );

// Provenance log (optional — may be null)
AdacProvenanceLog? log = await reader.ReadProvenanceLogAsync( "archive.adac" );

// Checksum manifest (optional — may be null)
AdacChecksumManifest? checksums = await reader.ReadChecksumsAsync( "archive.adac" );

// Region annotation (optional — may be null)
AdacRegionAnnotation? regions = await reader.ReadRegionAnnotationAsync(
    "archive.adac", "regions/master_0001.regions.json" );

// Edit pipeline (optional — may be null)
AdacEditPipeline? edits = await reader.ReadEditPipelineAsync(
    "archive.adac", "edits/master_0001.edits.json" );
```

Listing entries

```
IReadOnlyList<string> entries = await reader.ListContentsAsync( "archive.adac" );

foreach ( string entry in entries )
{
    Console.WriteLine( entry );
}
// manifest.json
// metadata/core.json
// master/master_0001.tif
// provenance/log.json
// provenance/checksums.json
```

Extracting files

```
// Extract a single entry by its container-relative path
await reader.ExtractFileAsync( "archive.adac", "metadata/core.json", @"C:\output\core.

// Extract a single master by ID
await reader.ExtractMasterAsync( "archive.adac", "master_0001", @"C:\output" );

// Extract all masters
await reader.ExtractAllMastersAsync( "archive.adac", @"C:\output\masters" );
```

IAdacValidator — Validating Containers

Validates a container against the ADAC 1.0 specification and returns structured findings.

Default validation (structure + checksums)

```
IAdacValidator validator = sp.GetRequiredService<IAdacValidator>();

AdacValidationResult result = await validator.ValidateAsync( "archive.adac" );

if ( result.IsValid )
{
    Console.WriteLine( "Container is valid." );
}
else
{
    foreach ( AdacValidationEntry entry in result.Errors )
    {
        Console.WriteLine( $"[{entry.Code}] {entry.Message}" );
    }
}
```

Custom validation options

```
AdacValidationOptions options = new()
{
    VerifyChecksums = false,            // Skip SHA-256 verification (faster)
    WarnOnMissingProvenanceLog = true,  // Warn if provenance/log.json is absent
    WarnOnMissingChecksums = true,      // Warn if provenance/checksums.json is absent
};

AdacValidationResult result = await validator.ValidateAsync( "archive.adac", options );
```

Inspecting findings by severity

```
// All findings
foreach ( AdacValidationEntry entry in result.Entries )
{
    Console.WriteLine( $"[{entry.Severity}] {entry.Code}: {entry.Message}" );
}

// Errors only
foreach ( AdacValidationEntry error in result.Errors )
{
    Console.WriteLine( $"ERROR: {error.Code} — {error.Message}" );
}

// Warnings only
foreach ( AdacValidationEntry warning in result.Warnings )
{
    Console.WriteLine( $"WARN: {warning.Code} — {warning.Message}" );
}
```

IAdacFixityService — Fixity and Checksums

Provides SHA-256 checksum computation, manifest generation, and container-level verification.

Verifying a container

```
IAdacFixityService fixity = sp.GetRequiredService<IAdacFixityService>();

AdacFixityReport report = await fixity.VerifyAsync( "archive.adac" );

Console.WriteLine( $"Valid:    {report.IsValid}" );
Console.WriteLine( $"Total:   {report.TotalFiles}" );
Console.WriteLine( $"Passed:  {report.VerifiedFiles}" );
Console.WriteLine( $"Failed:  {report.FailedFiles}" );
Console.WriteLine( $"Missing: {report.MissingFiles}" );

foreach ( AdacFixityMismatch m in report.Mismatches )
{
    if ( m.IsMissing )
    {
        Console.WriteLine( $"  MISSING: {m.Path}" );
    }
    else
    {
        Console.WriteLine( $"  MISMATCH: {m.Path}" );
        Console.WriteLine( $"    expected: {m.ExpectedChecksum}" );
        Console.WriteLine( $"    actual:   {m.ActualChecksum}" );
    }
}
```

Generating a checksum manifest

```
AdacChecksumManifest manifest = await fixity.GenerateManifestAsync( "archive.adac" );

foreach ( AdacFileChecksum file in manifest.Files )
{
    Console.WriteLine( $"{file.Checksum}  {file.Path}" );
}
```

Computing individual checksums

```
// From a file path
string hash = await fixity.ComputeChecksumAsync( @"C:\scans\photo.tif" );

// From a stream
using FileStream stream = File.OpenRead( @"C:\scans\photo.tif" );
string hash2 = await fixity.ComputeChecksumAsync( stream );
```

↑ Back to Top

Models

AdacManifest

Represents the container manifest (`manifest.json`) — the single source of truth describing the contents and structure of an ADAC container.

Property	Type	Description
`AdacVersion`	`string`	Specification version (e.g., `"1.0"`).
`Id`	`string`	Globally unique package identifier.
`CreatedOn`	`DateTimeOffset`	ISO-8601 creation timestamp.
`CreatedBy`	`string`	Software or person that created the container.
`Description`	`string`	Human-readable description of the artifact.
`Masters`	`List<AdacMasterEntry>`	Master file entries (at least one required).
`Derivatives`	`List<AdacDerivativeEntry>?`	Optional derivative file entries.
`Metadata`	`AdacMetadataReference`	Paths to metadata files.
`ExtensionData`	`Dictionary<string, JsonElement>?`	Unknown fields preserved for forward compatibility.

AdacMasterEntry

Property	Type	Description
`Id`	`string`	Unique master identifier (e.g., `"master_0001"`).
`File`	`string`	Container-relative path (e.g., `"master/master_0001.tif"`).
`Xmp`	`string?`	Optional path to XMP sidecar metadata.
`Regions`	`string?`	Optional path to region annotation file.
`Edits`	`string?`	Optional path to non-destructive edit pipeline.
`Encryption`	`AdacEncryptionDescriptor?`	Encryption metadata if the file is pre-encrypted ciphertext.

AdacDerivativeEntry

Property	Type	Description
`Id`	`string`	Unique derivative identifier.
`File`	`string`	Container-relative path.
`SourceMasterId`	`string`	Identifier of the source master.
`Purpose`	`string`	Purpose (e.g., `"access"`, `"thumbnail"`, `"web"`).
`Encryption`	`AdacEncryptionDescriptor?`	Encryption metadata if applicable.

AdacCoreMetadata

Represents the core metadata record (`metadata/core.json`) describing the digital artifact using Dublin Core-inspired fields.

Property	Type	Description
`Id`	`string`	Identifier matching the manifest `id`.
`Title`	`string?`	Human-readable title.
`Creator`	`string?`	Original creator (Dublin Core `dc:creator`).
`Subject`	`string?`	Subject keywords (Dublin Core `dc:subject`).
`Description`	`string?`	Content description.
`DateCreated`	`string?`	Original creation date (ISO-8601).
`Source`	`string?`	Source from which the artifact was derived.
`Format`	`string?`	Primary format (e.g., `"TIFF"`, `"WAV"`, `"DNG"`).
`Language`	`string?`	Content language (IETF BCP 47).
`Coverage`	`string?`	Geographic or temporal coverage.
`Tags`	`List<string>?`	Classification tags for categorization.
`Rights`	`AdacRights?`	Rights and licensing information.
`Technical`	`AdacTechnicalMetadata?`	Capture/creation technical details.
`Administrative`	`AdacAdministrativeMetadata?`	Lifecycle administrative details.
`Preservation`	`AdacPreservationMetadata?`	Preservation statistics.

AdacRights

Property	Type	Description
`Statement`	`string?`	General rights statement.
`Holder`	`string?`	Rights holder.
`License`	`string?`	SPDX identifier or free-text license.
`AccessRestrictions`	`string?`	Access restrictions (e.g., `"Embargoed until 2030"`).

AdacTechnicalMetadata

Property	Type	Description
`Scanner`	`string?`	Scanning device or capture system.
`Dpi`	`int?`	Scanning resolution (dots per inch).
`BitDepth`	`int?`	Bit depth (e.g., 8, 16, 24, 48).
`ColorSpace`	`string?`	Color space (e.g., `"sRGB"`, `"AdobeRGB"`).
`Duration`	`string?`	Audio/video duration.
`FrameRate`	`double?`	Video frame rate.
`SampleRate`	`int?`	Audio sample rate in Hz.
`PageCount`	`int?`	Page count for multi-page documents.

AdacAdministrativeMetadata

Property	Type	Description
`Repository`	`string?`	Archival repository name.
`AccessionNumber`	`string?`	Repository accession number.
`CatalogNumber`	`string?`	Catalog number.
`DigitizedBy`	`string?`	Person or system that performed digitization.
`DigitizedOn`	`string?`	ISO-8601 digitization timestamp.

AdacPackageDefinition

Defines everything needed to create a new ADAC container. Pass an instance to `IAdacContainerWriter.CreateAsync`.

Property	Type	Required	Description
`Id`	`string?`	no	Package identifier. Auto-generated GUID if omitted.
`Description`	`string`	no	Human-readable description.
`CreatedBy`	`string`	no	Creator identifier.
`CoreMetadata`	`AdacCoreMetadata`	yes	Core metadata for the artifact.
`Masters`	`List<AdacMasterSource>`	yes	Master files to include (at least one).
`Derivatives`	`List<AdacDerivativeSource>?`	no	Optional derivative files.
`ProfileSourcePaths`	`List<string>?`	no	Paths to domain-specific profile metadata files on disk.
`ProvenanceLog`	`AdacProvenanceLog?`	no	Optional provenance log to include.
`RegionAnnotations`	`List<AdacRegionSource>?`	no	Optional region annotations.
`EditPipelines`	`List<AdacEditSource>?`	no	Optional edit pipelines.

AdacMasterSource

Property	Type	Required	Description
`Id`	`string`	yes	Unique identifier (e.g., `"master_0001"`).
`SourceFilePath`	`string`	yes	Path to the source file on disk.
`ContainerPath`	`string?`	no	Desired container path. Auto-generated if omitted.
`XmpSourceFilePath`	`string?`	no	Path to an XMP sidecar file on disk.
`Encryption`	`AdacEncryptionDescriptor?`	no	Encryption metadata for pre-encrypted files.

AdacDerivativeSource

Property	Type	Required	Description
`Id`	`string`	yes	Unique identifier.
`SourceFilePath`	`string`	yes	Path to the source file on disk.
`ContainerPath`	`string?`	no	Desired container path. Auto-generated if omitted.
`SourceMasterId`	`string`	yes	Identifier of the source master.
`Purpose`	`string`	yes	Purpose (e.g., `"access"`, `"thumbnail"`).
`Encryption`	`AdacEncryptionDescriptor?`	no	Encryption metadata for pre-encrypted files.

AdacOperationProgress

Reported during container creation via `IProgress<AdacOperationProgress>`.

Property	Type	Description
`Phase`	`string`	Current phase (e.g., `"CopyingMasters"`, `"WritingChecksums"`).
`Current`	`int`	Current item index (1-based).
`Total`	`int`	Total items in the current phase.
`CurrentFile`	`string?`	Name or path of the file being processed.

AdacChecksumManifest and AdacFixityReport

AdacChecksumManifest

Represents the fixity manifest (`provenance/checksums.json`).

Property	Type	Description
`Algorithm`	`string`	Hash algorithm identifier (always `"sha256"` in ADAC 1.0).
`Files`	`List<AdacFileChecksum>`	File-checksum pairs.

Each `AdacFileChecksum` contains:

Property	Type	Description
`Path`	`string`	Container-relative file path.
`Checksum`	`string`	Lowercase hexadecimal SHA-256 hash.

AdacFixityReport

Returned by `IAdacFixityService.VerifyAsync`.

Property	Type	Description
`IsValid`	`bool`	`true` when all checksums match.
`TotalFiles`	`int`	Total files in the checksum manifest.
`VerifiedFiles`	`int`	Files that passed verification.
`FailedFiles`	`int`	Files with mismatched checksums.
`MissingFiles`	`int`	Files listed but absent from the container.
`Mismatches`	`List<AdacFixityMismatch>`	Details for each failure.

Each `AdacFixityMismatch` contains:

Property	Type	Description
`Path`	`string`	Container-relative path.
`ExpectedChecksum`	`string`	Hash from the stored manifest.
`ActualChecksum`	`string?`	Computed hash, or `null` if the file is missing.
`IsMissing`	`bool`	`true` when the file is absent.

AdacProvenanceLog

Represents the provenance log (`provenance/log.json`) — a chronological record of actions taken on the artifact.

Property	Type	Description
`Events`	`List<AdacProvenanceEvent>`	Chronological list of events.

Each `AdacProvenanceEvent` contains:

Property	Type	Description
`Id`	`string`	Unique event identifier.
`Type`	`string`	Event type (e.g., `"scan"`, `"import"`, `"edit"`, `"derivativeCreated"`).
`Timestamp`	`DateTimeOffset`	ISO-8601 timestamp.
`Actor`	`string`	Person or system that performed the action.
`Software`	`string?`	Software used (optional).
`Details`	`Dictionary<string, JsonElement>?`	Event-specific properties.

Example — creating a container with provenance:

```
AdacPackageDefinition package = new()
{
    CreatedBy = "my-app/1.0",
    CoreMetadata = new AdacCoreMetadata { Title = "Deed of sale" },
    Masters = { new AdacMasterSource { Id = "master_0001", SourceFilePath = "deed.tif"
    ProvenanceLog = new AdacProvenanceLog
    {
        Events =
        {
            new AdacProvenanceEvent
            {
                Id = "evt-001",
                Type = "scan",
                Timestamp = DateTimeOffset.UtcNow,
                Actor = "Digitization Lab",
                Software = "Epson Scan v5.0",
            },
        },
    },
};
```

AdacRegionAnnotation

Represents region annotations (`regions/<master>.regions.json`) that annotate spatial, temporal, or volumetric parts of a master artifact.

Property	Type	Description
`MediaId`	`string?`	Master identifier this annotation applies to.
`CoordinateSystem`	`string`	Coordinate system: `"pixel"`, `"timecode"`, or `"3d"`.
`Width` / `Height`	`int?`	Media dimensions (for pixel-based systems).
`Duration`	`string?`	Media duration (for time-based systems).
`Frames`	`int?`	Total frame count (for video).
`Regions`	`List<AdacRegion>`	Annotated regions.

Each `AdacRegion` contains:

Property	Type	Description
`Id`	`string`	Unique region identifier.
`Type`	`string`	Region type (e.g., `"boundingBox"`, `"timeSegment"`, `"point"`).
`Bounds`	`AdacRegionBounds?`	Spatial or temporal bounds.
`Label`	`string?`	Human-readable label.
`LinkedEntities`	`Dictionary<string, JsonElement>?`	Domain-specific linked entities.

AdacEditPipeline

Represents non-destructive edit instructions (`edits/<master>.edits.json`) stored as transformation parameters rather than altered pixels.

Property	Type	Description
`MediaId`	`string?`	Master identifier this pipeline applies to.
`PipelineVersion`	`string?`	Pipeline format version.
`CoordinateSpace`	`string`	Coordinate space for spatial parameters (default: `"pixel"`).
`ReferenceWidth` / `ReferenceHeight`	`int?`	Reference dimensions for scaling.
`Operations`	`List<AdacEditOperation>`	Ordered list of edit operations.

Each `AdacEditOperation` contains:

Property	Type	Description
`Id`	`string`	Unique operation identifier.
`Type`	`string`	Operation type (e.g., `"crop"`, `"rotate"`, `"colorAdjust"`).
`Parameters`	`Dictionary<string, JsonElement>?`	Operation-specific parameters.
`AppliesToDerivativeIds`	`List<string>?`	Derivatives produced using this operation.

AdacEncryptionDescriptor

Describes that a stored file was pre-encrypted before being placed into the container. ADAC never performs encryption or decryption — this is purely informational metadata.

Property	Type	Description
`Algorithm`	`string`	Encryption algorithm (e.g., `"AES-256-GCM"`).
`KeyId`	`string?`	Opaque key reference for the consumer's key management system.
`OriginalMediaType`	`string?`	MIME type of the unencrypted content (e.g., `"image/tiff"`).

Example — packaging a pre-encrypted master:

```
AdacMasterSource encryptedMaster = new()
{
    Id = "master_0001",
    SourceFilePath = @"C:\vault\photo.tif.enc",
    Encryption = new AdacEncryptionDescriptor
    {
        Algorithm = "AES-256-GCM",
        KeyId = "vault://keys/archive-2025",
        OriginalMediaType = "image/tiff",
    },
};
```

SHA-256 fixity checksums cover the ciphertext, which is the correct behavior — the container verifies integrity of what is stored, not the original plaintext.

↑ Back to Top

Validation

Validation Results

`AdacValidationResult` is returned by `IAdacValidator.ValidateAsync` and contains:

Property	Type	Description
`IsValid`	`bool`	`true` when no error-level findings exist.
`Entries`	`IReadOnlyList<AdacValidationEntry>`	All findings.
`Errors`	`IEnumerable<AdacValidationEntry>`	Error-level findings only.
`Warnings`	`IEnumerable<AdacValidationEntry>`	Warning-level findings only.

Each `AdacValidationEntry` contains:

Property	Type	Description
`Severity`	`AdacValidationSeverity`	`Info`, `Warning`, or `Error`.
`Code`	`string`	Stable machine-readable rule code (e.g., `"ADAC-010"`).
`Message`	`string`	Human-readable description.
`Path`	`string?`	Container-relative path involved, or `null`.

Validation Options

Property	Type	Default	Description
`VerifyChecksums`	`bool`	`true`	Verify SHA-256 checksums against actual content.
`WarnOnMissingProvenanceLog`	`bool`	`true`	Report missing `provenance/log.json` as a warning.
`WarnOnMissingChecksums`	`bool`	`true`	Report missing `provenance/checksums.json` as a warning.

↑ Back to Top

Constants

The `AdacConstants` class in the `Adac.Constants` namespace exposes all well-known paths and values defined by the ADAC 1.0 specification:

Constant	Value	Description
`Version`	`"1.0"`	Current ADAC specification version.
`FileExtension`	`".adac"`	Container file extension.
`MimeType`	`"application/vnd.adac.container"`	Registered MIME type.
`ManifestFileName`	`"manifest.json"`	Root manifest file.
`MasterDirectory`	`"master"`	Required master files directory.
`MetadataDirectory`	`"metadata"`	Metadata directory.
`CoreMetadataPath`	`"metadata/core.json"`	Required core metadata file.
`ProvenanceDirectory`	`"provenance"`	Provenance directory.
`ProvenanceLogPath`	`"provenance/log.json"`	Optional provenance log.
`ChecksumsPath`	`"provenance/checksums.json"`	Optional checksums manifest.
`DerivativesDirectory`	`"derivatives"`	Optional derivatives directory.
`RegionsDirectory`	`"regions"`	Optional regions directory.
`EditsDirectory`	`"edits"`	Optional edits directory.
`XmpDirectory`	`"metadata/xmp"`	Optional XMP metadata directory.
`ProfilesDirectory`	`"metadata/profiles"`	Optional domain-specific profiles directory.
`ChecksumAlgorithm`	`"sha256"`	Checksum algorithm identifier.

Additionally, `AdacConstants.JsonOptions` provides the shared `JsonSerializerOptions` (camelCase, indented, nulls omitted) used for all ADAC JSON serialization.

↑ Back to Top

Typical Workflows

Create, validate, and verify

```
// Create
AdacPackageDefinition package = new()
{
    CreatedBy = "my-app/1.0",
    CoreMetadata = new AdacCoreMetadata { Title = "Deed of sale, 1887" },
    Masters = { new AdacMasterSource { Id = "master_0001", SourceFilePath = "deed.tif"
};

await writer.CreateAsync( "deed.adac", package );

// Validate structure + checksums
AdacValidationResult result = await validator.ValidateAsync( "deed.adac" );

// Verify fixity independently
AdacFixityReport report = await fixity.VerifyAsync( "deed.adac" );
```

Read and inspect an existing container

```
AdacManifest manifest = await reader.ReadManifestAsync( "archive.adac" );
AdacCoreMetadata metadata = await reader.ReadCoreMetadataAsync( "archive.adac" );

Console.WriteLine( $"ADAC Version: {manifest.AdacVersion}" );
Console.WriteLine( $"Package ID:   {manifest.Id}" );
Console.WriteLine( $"Created:      {manifest.CreatedOn}" );
Console.WriteLine( $"Title:        {metadata.Title}" );
Console.WriteLine( $"Masters:      {manifest.Masters.Count}" );

foreach ( AdacMasterEntry master in manifest.Masters )
{
    Console.WriteLine( $"  {master.Id} → {master.File}" );
}
```

Extract all masters for offline processing

```
await reader.ExtractAllMastersAsync( "archive.adac", @"C:\working-copies" );
```

Create a container with derivatives and rich metadata

```
AdacPackageDefinition package = new()
{
    CreatedBy = "imaging-pipeline/2.0",
    CoreMetadata = new AdacCoreMetadata
    {
        Title = "Historical photograph",
        Format = "TIFF",
        Technical = new AdacTechnicalMetadata
        {
            Scanner = "Epson Expression 12000XL",
            Dpi = 600,
            BitDepth = 48,
            ColorSpace = "AdobeRGB",
        },
        Administrative = new AdacAdministrativeMetadata
        {
            Repository = "National Archives",
            AccessionNumber = "2025-001-0042",
            DigitizedBy = "Digitization Lab",
            DigitizedOn = "2025-07-15T10:30:00Z",
        },
    },
    Masters =
    {
        new AdacMasterSource
        {
            Id = "master_0001",
            SourceFilePath = @"C:\scans\photo.tif",
            XmpSourceFilePath = @"C:\scans\photo.xmp",
        },
    },
    Derivatives =
    [
        new AdacDerivativeSource
        {
            Id = "deriv_0001",
            SourceFilePath = @"C:\output\photo_access.jpg",
            SourceMasterId = "master_0001",
            Purpose = "access",
        },
    ],
```

```
};

await writer.CreateAsync( "photo.adac", package );
```

↑ Back to Top

Project Structure

```
Adac/
├── Adac.csproj                      # Class library (net10.0)
├── Constants/
│   └── AdacConstants.cs             # Well-known paths, versions, JSON options
├── Extensions/
│   └── ServiceCollectionExtensions.cs # AddAdacCore() DI registration
├── Models/
│   ├── AdacManifest.cs              # Manifest and master/derivative entries
│   ├── AdacCoreMetadata.cs          # Core metadata, rights, technical, admin
│   ├── AdacPackageDefinition.cs     # Package definition and source descriptors
│   ├── AdacChecksumManifest.cs      # Checksum manifest and fixity report
│   ├── AdacProvenanceLog.cs         # Provenance log and events
│   ├── AdacRegionAnnotation.cs      # Region annotations and bounds
│   ├── AdacEditPipeline.cs          # Non-destructive edit pipelines
│   └── AdacEncryptionDescriptor.cs  # Pre-encryption metadata
├── Services/
│   ├── IAdacContainerReader.cs      # Reader interface
│   ├── AdacContainerReader.cs       # Reader implementation
│   ├── IAdacContainerWriter.cs      # Writer interface
│   ├── AdacContainerWriter.cs       # Writer implementation
│   ├── IAdacValidator.cs            # Validator interface
│   ├── AdacValidator.cs             # Validator implementation
│   ├── IAdacFixityService.cs        # Fixity service interface
│   ├── AdacFixityService.cs         # Fixity service implementation
│   └── HashingStream.cs            # Internal stream wrapper for hashing
└── Validation/
    └── AdacValidationResult.cs      # Validation results, entries, options
```

↑ Back to Top

License

↑ Back to Top