# Supporting Resources and Valid Content

# Baselines

FedRAMP's baselines are available in OSCAL XML, JSON and YAML formats on the OSCAL Foundation's fedramp-resources GitHub repository.

The OSCAL Foundation is making FedRAMP baselines available both as OSCAL *profiles* and as pre-processed *resolved profile catalogs*. While organizations can initially use the *resolved profile catalogs* to reduce complexity and get started more quickly, full profile resolution is required to handle control overlays and multiple frameworks. *Profiles* are authoritative. *Resolved Profile Catalogs* are for convenience only.

See the Concept of Operations below.

## Quick Start (MVP/CORE)

Get started quickly by skipping profile resolution processing and instead using "resolved profile" catalogs of the FedRAMP baseliens.

Although OSCAL offers a great deal of flexibility with baselines and overlays, if you need to get started quickly and just want a single OSCAL file with the controls for your baseline.

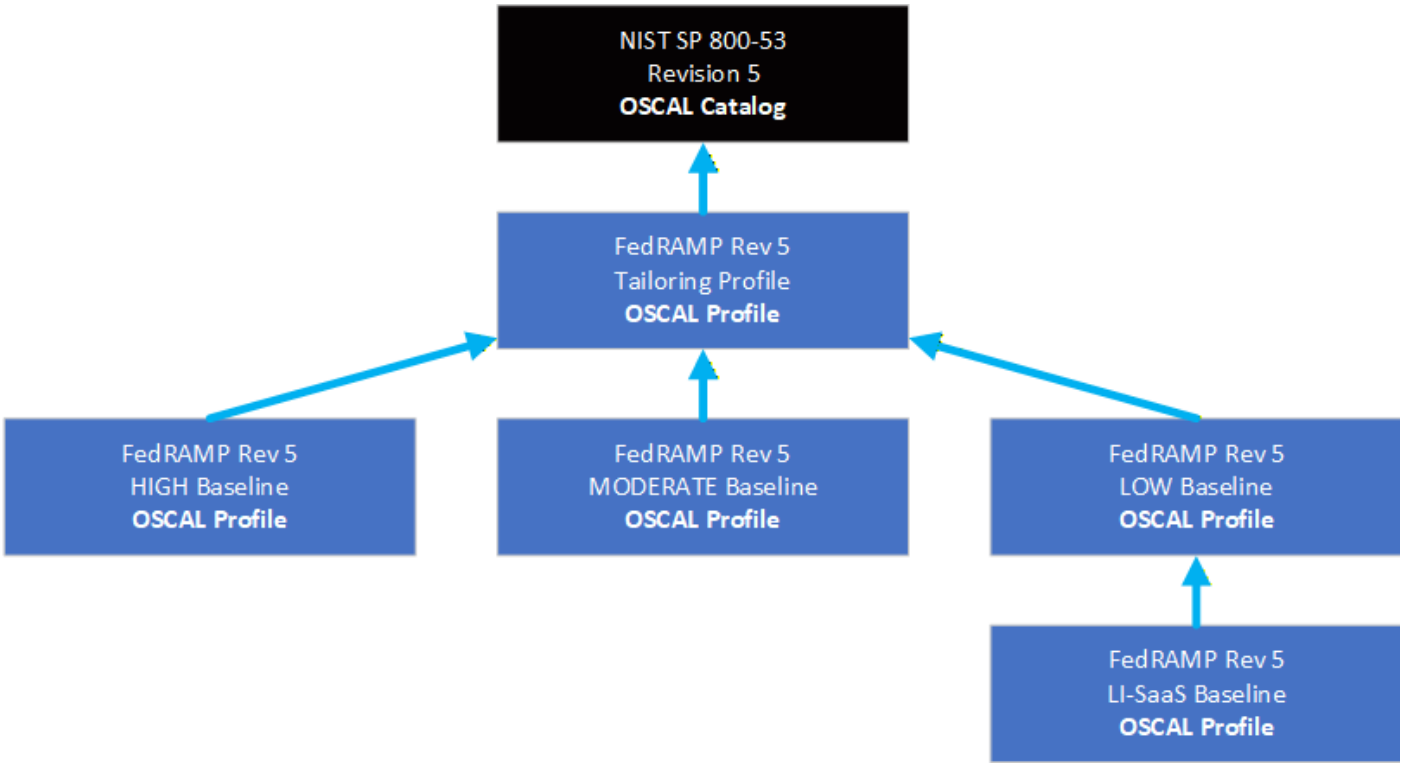
The following OSCAL "resolved profile" catalogs are exactly what you need:

- FedRAMP HIGH Baseline (OSCAL Catalog) [ XML | JSON | YAML ]
- FedRAMP MODERATE Baseline (OSCAL Catalog) [ XML | JSON | YAML ]
- FedRAMP LOW Baseline (OSCAL Catalog) [ XML | JSON | YAML ]
- FedRAMP LI-SaaS Baseline (OSCAL Catalog) [ XML | JSON | YAML ]

## OSCAL Tailoring and Overlays

OSCAL is designed to be referential. It enables tailoring of controls and the ability to overlay controls. The FedRAMP OSCAL profiles are available for these more complex scearios.

The following referential structure is used:




- A single FedRAMP Rev 5 tailoring profile imports the NIST SP 800-53 Rev 5 catalog. FedRAMP control tailoring that applies to all baselines is performed here.
- High, Moderate, and Low FedRAMP Rev 5 Profiles each import the FedRAMP Tailoring Profile. Each also identifies the controls for that baseline and include any baseline-speicifc control tailoring.
- Low-Impact SaaS is a special FedRAMP tailoring of the FedRAMP Low baseline. It imports the FedRAMP Low Profile and tailors it further.

The "resolved profile" catalogs at the top of this page are the result of processing the control selection and tailoring represented here.

## Available OSCAL Catalog and Profiles

The following OSCAL catalogs and profiles are available:

- NIST SP 800-53, Revision 5 (OSCAL Catalog) [ XML | JSON | YAML ]
- FedRAMP Tailoring Profile (OSCAL Profile) [ XML | JSON | YAML ]
- FedRAMP HIGH Baseline (OSCAL Profile) [ XML | JSON | YAML ]
- FedRAMP MODERATE Baseline (OSCAL Profile) [ XML | JSON | YAML ]
- FedRAMP LOW Baseline (OSCAL Profile) [ XML | JSON | YAML ]
- FedRAMP LI-SaaS Baseline (OSCAL Profile) [ XML | JSON | YAML ]

---

## Concept of Operations

OSCAL *catalogs* are used to define controls. OSCAL *profiles* are used to establish control baselines, tailor controls, handle control overlays, and aggregate controls from multiple compliance frameworks.

OSCAL tools SHOULD process profiles as the authoritative representation for baselines, tailored controls, overlays, and aggregated frameworks imposed on a specific system.

## Resolved Profile Catalogs

It is possible to pre-process OSCAL profiles and catalogs into a cached *resolved profile catalog*. This result is represented as an OSCAL catalog, using the same syntax. The OSCAL Foundation has done this for the FedRAMP baselines.

| Benefits | Tradeoffs |
| --- | --- |
| Allows organizations to initially skip profile processing and focus on other aspects of OSCAL adoption | Overlays and multiple frameworks are not possible. The cached content can become stale. |

> Organizations electing to start their OSCAL journey using resolved profilie catalogs will need to add proper profile resolution to your roadmap for control overlays and multi-framework processing.

---

# Examples

This content uses YAML for examples. All examples are derived from complete example OSCAL content, which is available in all three OSCAL formats and published in the OSCAL Foundation's fedramp-resources GitHub repository:

| FedRAMP OSCAL Artifact | | | | Status |
|---|---|---|---|---|
| System Security Plan (SSP) | XML | JSON | YAML | *In Progress* |
| Plan of Action and Milestones (POA&M) | | | | *Next* |
| Security Assessment Plan (SAP) | | | | *Phase 3* |
| Security Assessment Report (SAR) | | | | *Phase 3* |

Additional examples will be published here as they become available.

# Validating Content

The adoption of standardized, machine-readable security data requires a rigorous approach to ensuring data integrity across various layers of complexity. By implementing a systematic validation framework, organizations can transition from manual document reviews to automated, high-assurance compliance monitoring.

# The Tiered Validation Model

Validating OSCAL content is a tiered process that ensures data integrity from basic file structure to complex compliance requirements. For organizations following the Retrofit FedRAMP adoption path, validation requirements evolve as you move from initial MVP stages toward the full Target State.

---

## Layers of OSCAL Validation

To ensure high-quality authorization package data, OSCAL documents must pass through four distinct validation layers. Each layer builds upon the previous one, and a failure at any stage risks rendering the document invalid for FedRAMP submission.

| Layer | Validation Type | Description |
|---|---|---|
| **Well-Formed** | File Format Valid | Confirms the XML, JSON, or YAML file follows the fundamental rules of its format. If a tool can parse the file without syntax errors, it is considered well-formed. |
| **OSCAL Syntax** | OSCAL Schema Valid | Verifies that the file uses only the names, structures, and values defined in the official NIST OSCAL schemas. This ensures the document aligns with the core model (e.g., SSP, SAP, or SAR). |
| **Core OSCAL** | Metaschema Valid | Uses Metaschema constraints to validate the internal data logic. This checks relationships within the document, ensuring that UUIDs match and cross-references between elements are functionally sound. |
| **FedRAMP OSCAL** | FedRAMP Valid | Enforces FedRAMP-specific business rules via an additional layer of constraints. This ensures the content meets technical requirements specific to the FedRAMP PMO, such as required metadata fields and specific parameter formatting. |

---

## Validation and the Adoption Path

Validation is not a "one size fits all" requirement at the start of your transition.

Depending on your current position in the Retrofit Adoption Path, your validation focus will shift:

- **MVP Tier (Minimum Viable Product):** Focus is primarily on **Well-Formed** and **OSCAL Syntax** validation. At this stage, the priority is successfully converting legacy data into an OSCAL-compliant structure.
- **Intermediate Tier:** Includes **Core OSCAL** validation. Relationships between components and controls must be logically sound as you begin to automate the mapping of your security architecture.
- **Advanced Tier:** Introduces **FedRAMP** validation. As the document matures, it must begin meeting the specific programmatic constraints required for official PMO review.
- **Target State:** Continuous validation across all four layers. In this state, you should automatically reject any changes that fail any validation layer, ensuring your "Source of Truth" remains submission-ready at all times.

---

## Recommended Tooling

Validation is most effective when integrated into the content development workflow rather than treated as a final check. The tools and resources listed below are highly recommended for OSCAL content validation.

- **Metaschema OSCAL CLI:** The Metaschema-Framework OSCAL-CLI is the primary tool for performing schema and Metaschema validation.
- **FedRAMP Automation Suite (decommissioned):** Specific validators created by the FedRAMP PMO to check against their unique business rules for Rev5 systems. This work is no longer maintained by FedRAMP but demonstrates how the OSCAL-CLI and external constraints can be used to validate OSCAL content against FedRAMP validation rules.
- **IDE Extensions:** Using integrated development environments (IDE) or similar editors with schema-aware extensions provides real-time feedback during the authoring process.
- **AI-Assisted Remediation:** Large Language Models (LLMs) and specialized AI agents can be used to automate validation of OSCAL documents, interpret complex validation error logs, and quickly generate corrected JSON/YAML snippets to remediate repetitive formatting issues.

# Validating FedRAMP Content with OSCAL CLI

## Get Started

The oscal-cli is an open source command-line utility designed to help developers and security professionals interact with OSCAL. To get started, follow the installation instructions from the OSCAL-CLI GitHub "README" page. Once installed, you can use the tool to perform core tasks such as validating OSCAL JSON, YAML, or XML files against the official schemas, converting between formats, and resolving "profile" documents into comprehensive "catalog" views. For those working in containerized environments, the tool is also available as a Docker image, allowing for easy integration into automated CI/CD compliance pipelines.

## Commands

The following are a brief summary of the most commonly used oscal-cli commands when working with FedRAMP OSCAL documents.

### Validate

Checks that the specified OSCAL file is well-formed and valid per OSCAL model and any additional specified constraints.

```
oscal-cli validate <file> -c <constraint-file> [options]
```

### Convert

Converts an OSCAL source file to the specified file format (e.g., XML, JSON, or YAML).

```
oscal-cli convert --to=FORMAT <source-file-or-URL> <destination-file>
```

### Profile Resolution

Creates a new tailored OSCAL catalog by combining baseline imported controls and applying any modifications, as specified in the source profile.

```
oscal-cli resolve-profile <profile-file>
```

### Metaschema Validate

Validates that a Metaschema file is well-formed and valid per the specified constraint definitions.

```
oscal-cli metaschema validate <file> -c <style-guide> [options]
```

# FedRAMP Validation Examples

## Validating SSP Against FedRAMP Constraints

```
oscal-cli validate path/to/ssp.xml \
  -c path/to/fedramp-external-allowed-values.xml \
  -c path/to/fedramp-external-constraints.xml
```

# Understanding Validation Errors

## Common Error Messages

1. UUID Warning:

```
[WARNING] [.../protocol[1]] It is a best practice to provide a UUID.
```

**Resolution**: Add a unique UUID to the specified element.

2. Constraint Violations:

- Invalid allowed values
- Missing required fields
- Pattern matching failures

# SARIF Output Integration

SARIF (Static Analysis Results Interchange Format) provides detailed validation results viewable in VS Code.

## Generate SARIF Output

```
oscal-cli validate path/to/file.xml \
  -c path/to/constraints.xml \
  --sarif-include-pass \
  -o results.sarif.json
```

## SARIF Features

- Detailed error locations
- Stack traces for debugging
- Pass/fail results for each rule
- VS Code integration for visual feedback

# Best Practices

1. Always validate at both levels:
   - Basic OSCAL schema validation
   - FedRAMP constraints validation
2. Always validate against both allowed values and external constraints
3. Use SARIF output for detailed analysis:

   ```
   oscal-cli validate path/to/document.xml \
     -c path/to/constraints.xml \
     --sarif-include-pass \
     -o results.sarif.json
   ```

4. Address all warnings, even if they don't cause validation failures
5. Keep constraint files updated with latest FedRAMP requirements