Community Contribution Series

OSCAL-Pydantic: A python library for OSCAL
Credentive Security
Background
Oscal-Pydantic

https://github.com/RS-Credentive/oscal-pydantic/

• An API for creating and manipulating OSCAL data models in Python
• Provides high-fidelity schemas of all OSCAL data elements
• Strives to be the reference implementation of OSCAL in Python

`pip install oscal-pydantic`
Building blocks

• Python (https://python.org)
  • Popular scripting/programming language
    • #2 behind Javascript on Github
  • Applications include
    • Web programming (server side)
    • Scientific Computing (Big Data/AI)
  • Strict, Dynamically typed language

• Pydantic (https://docs.pydantic.dev/latest/)
  • Most widely used data validation library for Python
  • Leverages type hints to provide strict, static typing
  • Supports serialization to/from JSON
OSCAL Schema

https://pages.nist.gov/OSCAL-Reference/models/

- Built on top of NIST Metaschema
  - https://pages.nist.gov/metaschema/
  - a common, format-agnostic modeling framework supporting schema, code, and documentation generation
OSCAL-Pydantic v1

- Inspired by Compliance Trestle
  - https://github.com/IBM/compliance-trestle
- Dynamically generated from JSON Schema
  - OSCAL JSON Schema → Datamodel-code-generator → Pydantic Models
  - Hand tweaked to eliminate some issues with JSON Schema translation
    - JSON “format” vs Regex
    - Unicode Regex – e.g. “(\p{L}|\p{L}|\p{N}|[\-\_])*”
- Outcome
  - Lightweight library to support generation and validation of OSCAL Data and import JSON objects
Example OSCAL data element

The value **may be locally defined**, or one of the following:

- **SHA-224**: The SHA-224 algorithm as defined by NIST FIPS 180-4.
- **SHA-256**: The SHA-256 algorithm as defined by NIST FIPS 180-4.
- **SHA-384**: The SHA-384 algorithm as defined by NIST FIPS 180-4.
- **SHA-512**: The SHA-512 algorithm as defined by NIST FIPS 180-4.
- **SHA3-224**: The SHA3-224 algorithm as defined by NIST FIPS 202.
- **SHA3-256**: The SHA3-256 algorithm as defined by NIST FIPS 202.
- **SHA3-384**: The SHA3-384 algorithm as defined by NIST FIPS 202.
- **SHA3-512**: The SHA3-512 algorithm as defined by NIST FIPS 202.

**hash**

<table>
<thead>
<tr>
<th>DESCRIPTION</th>
<th>A representation of a cryptographic digest generated over a resource using a specified hash algorithm.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>value</strong></td>
<td><strong>string</strong></td>
</tr>
<tr>
<td></td>
<td>[0 or 1]</td>
</tr>
<tr>
<td></td>
<td>Switch to XML</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**algorithm**

<table>
<thead>
<tr>
<th>DESCRIPTION</th>
<th>The digest method by which a hash is derived.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>value</strong></td>
<td><strong>string</strong></td>
</tr>
<tr>
<td></td>
<td>[0 or 1]</td>
</tr>
<tr>
<td></td>
<td>Switch to XML</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**value**

<table>
<thead>
<tr>
<th>DESCRIPTION</th>
<th>This property provides the (nominal) value for this object as a whole.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>value</strong></td>
<td><strong>string</strong></td>
</tr>
<tr>
<td></td>
<td>[0 or 1]</td>
</tr>
<tr>
<td></td>
<td>Switch to XML</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
class Hash(BaseModel):
    class Config:
        extra = Extra.forbid

    algorithm: Annotated[
        str,
        Field(
            description="Method by which a hash is derived",
            regex="^\S(.\S)\S?$",
            title="Hash algorithm",
        ),
    ]

    value: str
Issue: Machine code is not for humans

• Autogenerated schemas are tough to read and use
  • Lots of Root Models
  • A lot of repetition

• Difficult to extend/customize
Issue: Inherited JSON Schema limitations

• Constraints are limited to attribute values (regex) or “formats”
• No way to define relationships between attributes
  • IF “algorithm” == “SHA-224”
    • THEN ”value” must be a 28 character string
    • AND only comprised of the characters 0-9, a-f, or A-F
Osca!-Pydantic v2

Back to the drawing board
Oscal-Pydantic v2 Approach

• Leverage Pydantic v2
  • 4x – 50x faster than Pydantic v1 (~17x in general)

• Hand produced
  • Less Repetition
  • Designed for humans to extend and customize
  • Closer alignment with underlying metaschema
  • (In Progress) Support for all validation rules
OSCAL-Pydantic v2 Example

```python
OscalString = Annotated[str, constr(pattern=r'^\s\S+')]  

class Hash(base.OscalModel):
    algorithm: datatypes.OscalString | None = Field(
        description="""
        Method by which a hash is derived <...>
        """
    , default=None,
    pattern="^SHA3?-\d{2,4}$",
    )

    value: datatypes.OscalString | None = Field(
        description="""
        The value of the hash
        """
    , default=None,
    )
```
@model_validator(mode="after")

def validate_hash_for_algorithm(self):
    if self.algorithm is not None and self.value is None:
        raise ValueError("Hash Algorithm specified without Value")
    elif self.algorithm == "SHA-224" or self.algorithm == "SHA3-224":
        if len(self.value) == 28 and self.value_is_hex():
            return self
        else:
            raise ValueError("Hash value length or contents do not match algorithm")
    elif self.algorithm == "SHA-256" or self.algorithm == "SHA3-256":
        ...
    elif self.algorithm == "SHA-384" or self.algorithm == "SHA3-384":
        ...
    elif self.algorithm == "SHA-512" or self.algorithm == "SHA3-512":
        ...
    else:
        return self

def value_is_hex(self) -> bool:
    # Quick trick to check if a string is only HEX
    # try to convert it to an int.
    # If it doesn't work, there's a bad character in there.
    try:
        int(self.value, 16)
        return True
    except ValueError:
        return False
A tour of OSCAL-Pydantic v2
Oscal-Pydantic core modules

- oscal_pydantic.core.datatypes
  - core Metaschema datatypes
- oscal_pydantic.core.base.OscalModel
  - Subclass of pydantic.BaseModel
  - Common Field Alias Generator
    - Json attributes (“-”) to snake_case (“_”)
    - Can’t use “class” as a field (reserved word)
  - Override “model_dump_json” method
    - Exclude null, always use alias, pretty print
  - Maybe common content validator code?
Oscal-Pydantic core modules

- oscal-pydantic.core.common
  - Common OSCAL elements that appear in many parts of models
- oscal-pydantic.core.properties
  - Implements documented OSCAL properties
  - Implements full set of constraints
  - User Extensible
  - UNDER CONSTRUCTION
Oscal-Pydantic modules

- oscal_pydantic.catalog <- Now
- oscal_pydantic.system_security_plan
- oscal_pydantic.profile
- oscal_pydantic.component_definition
- oscal_pydantic.assessment_plan
- oscal_pydantic.assessment_results
- oscal_pydantic.plan_of_action_and_milestones
OSCAL Pydantic in Action
Help Wanted
Next Steps

• Fix Properties
• Finish Catalog
• Develop elements for Test Cases
Help Wanted: Properties

• Properties are very special parts of the OSCAL specification
  • Complex validation rules
  • Numerous different types of properties
  • Designed to be extensible

• OSCAL Pydantic must also provide full validation for all the various properties, and must support extension

• Currently working to identify the best way to enforce validation rules in an easily extensible way.
Help Wanted: Test Cases

• Integration testing is critical to maintaining a high-quality implementation
• Testing requires a library of valid and invalid artifacts
• Needed for every project implementing OSCAL regardless of language or domain.
• Suggestion: Should test cases be a separate project maintained on behalf of the community?