OSCAL Community Contribution Series:
A Developer’s view of OSCAL

Experiences and recommendations for implementing OSCAL Libraries
• User’s view vs. Developer’s view
• What is an OSCAL library for?
• Types of Objects in Oscal-Pydantic
• Oscal-Pydantic Class Diagram overview
• Recommendations for implementing OSCAL in any (Object Oriented) language
User’s view versus Developer’s view

**UX** (User Experience) creates a pleasant experience for **non-technical** folks trying to solve **business problems**.

**DX** (Developer Experience) creates a pleasant experience for **technical** folks trying to develop tooling to solve **user problems**.
User fights the problem, not the interface.

Developer fights the problem, not the API.
What are the features of a good API?

• Easy to do the most common things
• The right way is the easiest way
• Idiomatic
• Features are discoverable
• Features are implemented consistently
• Minimize boilerplate (DRY)
What is an OSCAL API for?
What problem is an OSCAL API solving?

• Reduce cognitive burden of maintaining mental copies of the OSCAL specification
• Easily focus on the part of the specification that you need for the problem at hand
• Produces OSCAL data that is valid and well formed.
• OSCAL imported from external sources is presented idiomatically
• Users can easily extend the specification with minimal impact on compatibility.
<table>
<thead>
<tr>
<th>Machine Generated Code</th>
<th>Handwritten Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>✓ Tracks the specification closely</td>
<td>X  Implements as much of the spec as the author needs</td>
</tr>
<tr>
<td>✓ Can always be up to date</td>
<td>X  Updated occasionally</td>
</tr>
<tr>
<td>✓ Can implement multiple versions of the specification</td>
<td>X  Implements the versions the author is interested in</td>
</tr>
<tr>
<td>X  Limited to Core Specification</td>
<td>✓  Can be designed for extensibility</td>
</tr>
<tr>
<td>X  Limited by the quality of the inputs</td>
<td>✓  Not dependent on limitations or structure of underlying specifications</td>
</tr>
<tr>
<td>X  Produces verbose, difficult code</td>
<td>✓  Code can be carefully tuned for readability</td>
</tr>
</tbody>
</table>
Oscal-Pydantic
A pythonic, type-hinted library for importing or generating OSCAL documents.

- Implemented in a “dataclass” like format, using Pydantic
- Hand-written to optimize the developer experience for encoding and extending OSCAL.
Oscal-Pydantic Overview

- **Encapsulate Behavior**
  - OscalModel
  - BaseProperty
  - BasePart
  - ...

- **Define Structure**
  - OscalProperty
  - RMFProperty
  - OscalPart
  - datatypes

- **Impose Constraints**
OscarModel: Encapsulate Shared Behavior

- Encapsulates core validation Logic
- Defines basic behaviors common to all models
  - How to export JSON
  - How to translate attribute value names
- Most objects extend OscarModel directly
Datatypes

• Defines basic constraints for metaschema datatypes
• List Attributes of a Model
• Define basic type information

class BaseProperty(base.OscalModel):
    # NOTE: This generic Property class should be extended to
    # provide constraints on value name
    name: datatypes.OscalToken = Field()
    uuid: datatypes.OscalUUID | None = Field(default=None)
    ns: datatypes.OscalUri = Field(
        default=datatypes.OscalUri(
            "http://csrc.nist.gov/ns/oscal"
        )
    )
    value: datatypes.OscalString = Field()
    prop_class: datatypes.OscalToken | None = Field(default=None)
    remarks: datatypes.OscalMarkupMultiline | None = Field(default=None)
Impose Constraints

- Cardinality Constraint
- Value Constraints
- “Type” constraints
- Deprecation Constraints
- Uniqueness Constraints
Cardinality Constraints

- How many attributes can appear
  - Usually 0..1, 1, or 0..inf

# Cardinality 1
uuid: datatypes.OscalUUID

# Cardinality 0..1
description: datatypes.OscalMarkupMultiline | None

# Cardinality 0..inf
props: list[properties.BaseProperty] | None
• Possible values of attributes
• Expressed as single values or sets of values
• Multiple Constraints can be expressed
• Within a constraint - AND
• Between constraints – OR
• Watch for “may be locally defined”, or allow-
  other="yes"

```python
class OscalResourceProperty(OscalBaseProperty):

    def __init__(self, properties):
        super().__init__(properties)

    def get_value(self, name):
        return self.properties.get(name)

    def set_value(self, name, value):
        self.properties[name] = value

    @property
    def name(self):
        return self.properties.get("name")

    @name.setter
    def name(self, value):
        self.properties["name"] = value

    @property
    def datatypes(self):
        return self.properties.get("datatypes")

    @datatypes.setter
    def datatypes(self, value):
        self.properties["datatypes"] = value

    @property
    def OscalToken(self):
        return self.properties.get("OscalToken")

    @OscalToken.setter
    def OscalToken(self, value):
        self.properties["OscalToken"] = value

    @property
    def OscalString(self):
        return self.properties.get("OscalString")

    @OscalString.setter
    def OscalString(self, value):
        self.properties["OscalString"] = value

    @property
    def version(self):
        return self.properties.get("version")

    @version.setter
    def version(self, value):
        self.properties["version"] = value

    @property
    def type(self):
        return self.properties.get("type")

    @type.setter
    def type(self, value):
        self.properties["type"] = value

    @property
    def logo(self):
        return self.properties.get("logo")

    @logo.setter
    def logo(self, value):
        self.properties["logo"] = value

    @property
    def image(self):
        return self.properties.get("image")

    @image.setter
    def image(self, value):
        self.properties["image"] = value

    @property
    def screen_shot(self):
        return self.properties.get("screen-shot")

    @screen_shot.setter
    def screen_shot(self, value):
        self.properties["screen-shot"] = value

    @property
    def published(self):
        return self.properties.get("published")

    @published.setter
    def published(self, value):
        self.properties["published"] = value
```

AND OR OR
“Type” Constraints

• NOTE: Types do not exist in the Metaschema specification
  • A “type” encapsulates a specific set of value constraints
• Define Possible types of attributes

```python
class BasePart(base.OscalModel):
    allowed_field_types: list[base.AllowedFieldTypes] = [  
        {  
            "props": [  
                properties.OscalPartProperty,  
                properties.OscalAssessmentMethodProperty,  
                properties.RmfAssessmentMethodProperty,  
            ],  
        },  
    ]
```
• Raise a warning in the event of a valid but deprecated value

```python
class OscalControlProperty(OscalBaseProperty):
    def capitalized_withdrawn_deprecated(cls, value: str) -> datatypes.OscalToken:
        # raise a deprecationwarning if value is capitalized
        if type(value) == str and value == "Withdrawn":
            warnings.warn(
                "Withdrawn' is a deprecated property value for Control. Use 'withdrawn' instead",
                DeprecationWarning,
            )
        return value
```
Uniqueness Constraint

- Prevent duplicate values in some cases

```python
class Resource(base.OscalModel):

def unique_rlink(self) -> Resource:
    if self.rlinks is not None:
        links_counter = Counter([rlink.href for rlink in self.rlinks])
        duplicates = [item for item, count in links_counter.items() if count > 1]
        if len(duplicates) > 0:
            raise ValueError("Duplicate rlinks in %s: %s", self.uuid, duplicates)
    return self
```
• Some kinds of data have unique constraints
  • Example: Hash
    • algorithm + value
    • value must be valid for the algorithm

```python
if self.algorithm == "SHA-224" or self.algorithm == "SHA3-224":
    if len(self.value) == 28 and self.value_is_hex():
        # value is okay
    else:
        raise ValueError("Hash value length or contents do not match algorithm")
```
Summary
• Identify the key features of the API
  • Extensibility?
• Look at the Metaschema or official documentation
  • JSON Schema is incomplete!
• Don’t be afraid to start by hand coding