An Introduction to the GEMD Data Model

(based on `gemd-python` v1.8.1)

Maggie Eminizer
Institute for Data Intensive Engineering and Science (IDIES)
Johns Hopkins University

Outline

Core Concepts

- GEMD and how we use it
- Object categories
- Attribute categories
- Value types
- Templates, Specs, and Runs

Further Details

- Attribute templates & Bounds
- Attributes
- Object Templates, Specs & Runs
- Other misc. points throughout the above

Practical Concerns

- `gemd-python` inheritance
- DictSerializable & JSON Serialization
- Some helpful functions
- Further resources

Core Concepts

What is GEMD?

- “Graphical Expression of Materials Data”
- Open-source, developed by Citrine Informatics
- Link materials w/ processes that created them and measurements performed on them
  - Arbitrary level of detail at each stage
  - Capture material provenance in a graph-based way
  - Separately describe all things that can be done, intent to do a thing, and outcome of doing the thing
- Python implementation
- JSON-based serialization
- Working to apply to PARADIM and DMREF projects

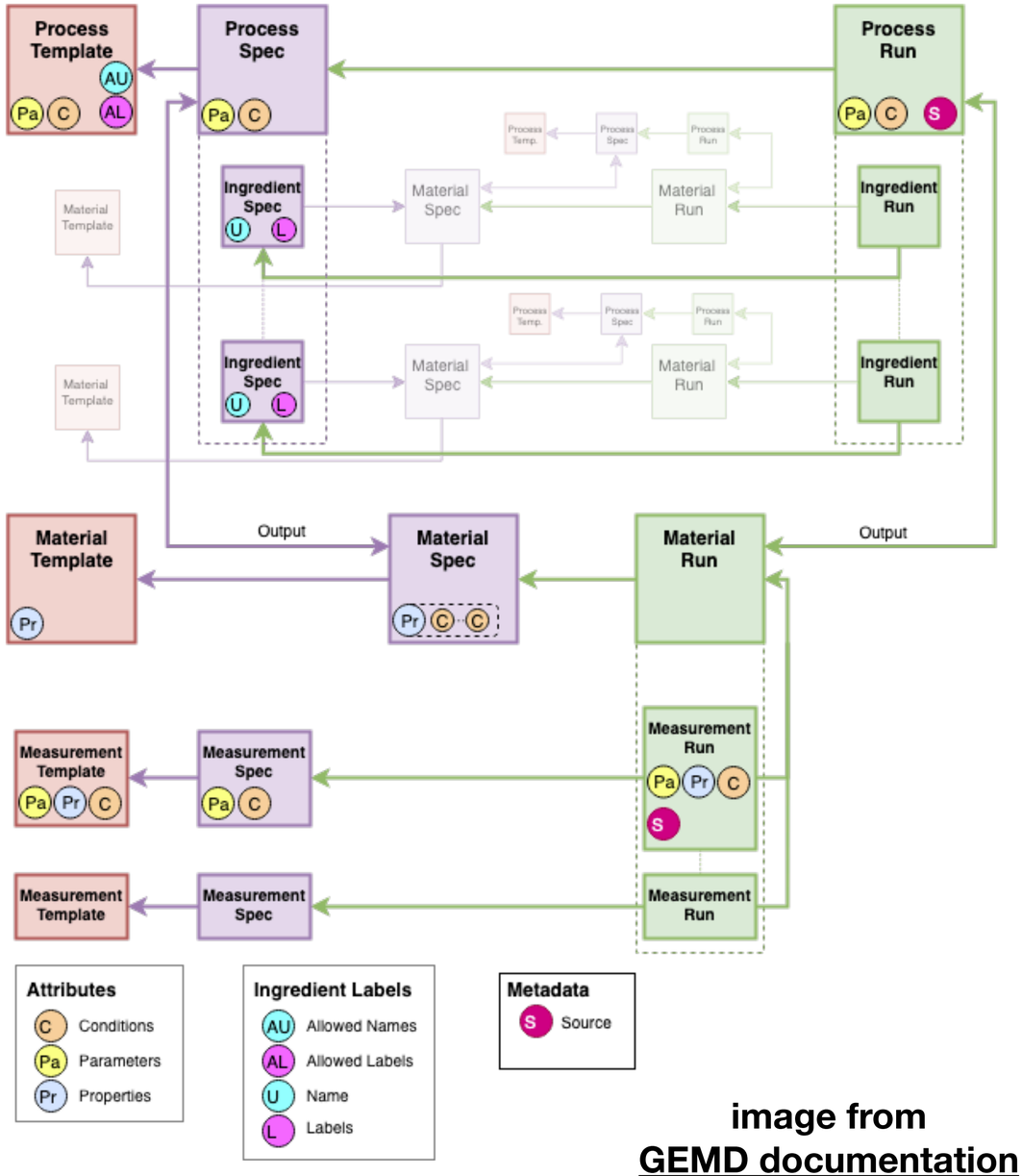
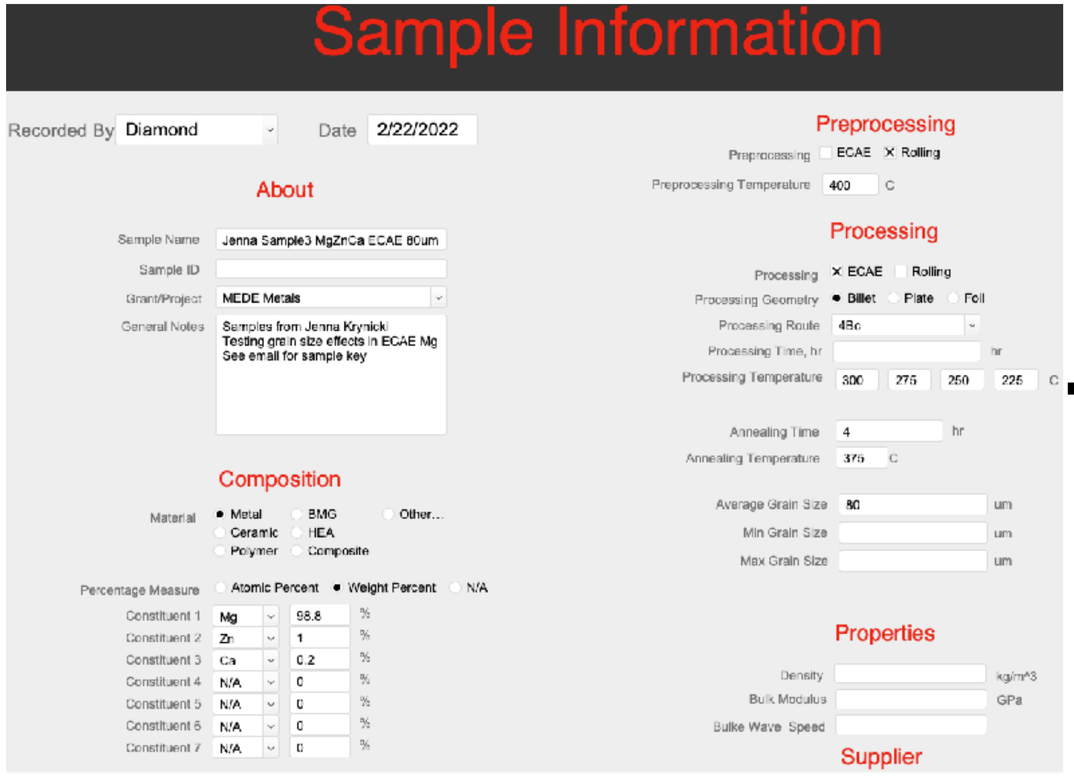
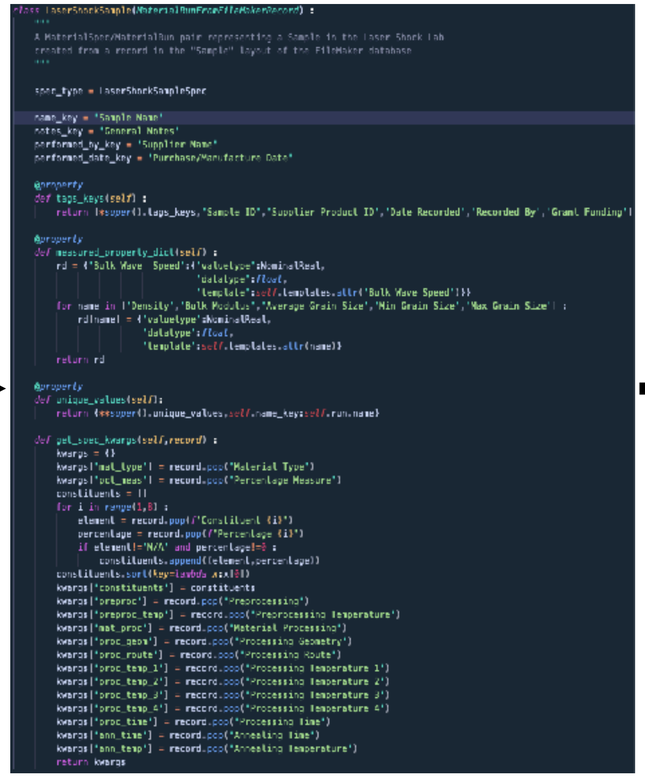


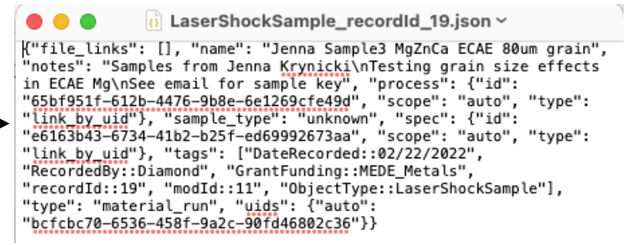
image from GEMD documentation

How Are We Using GEMD?

- KT Ramesh's Laser Shock Lab FileMaker database -> GEMD JSON/graph







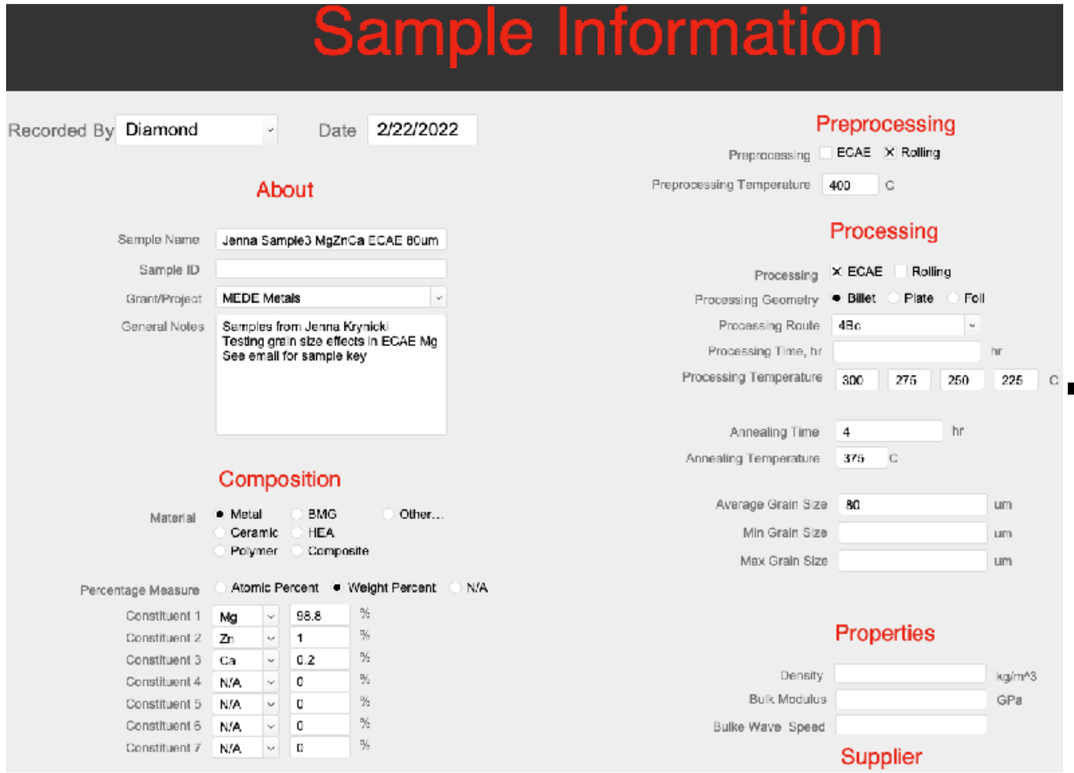
How Are We Using GEMD?

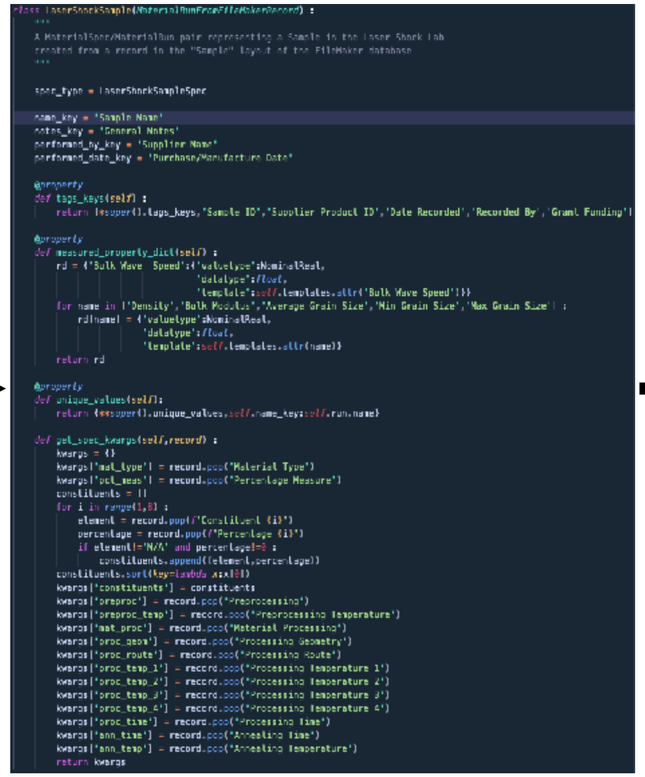
- KT Ramesh's Laser Shock Lab FileMaker database -> GEMD JSON/graph
- San Diego Supercomputer Center
  - Query models by storing object details and graph of relationships in a "polystore"
- Need more examples of GEMD model instances for PARADIM specifically

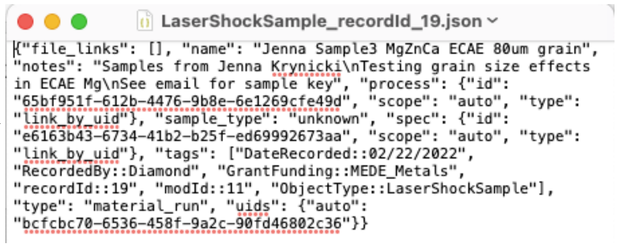
PHYSICAL REVIEW MATERIALS 3, 094407 (2019)

New kagome prototype materials: discovery of KV_3Sb_5, RbV_3Sb_5, and CsV_3Sb_5

Brenden R. Ortiz,[1,2,*] Lídia C. Gomes,[3,4] Jennifer R. Morey,[5] Michal Winiarski,[5,6] Mitchell Bordelon,[2] John S. Mangum,[1] Iain W. H. Oswald,[7] Jose A. Rodriguez-Rivera,[8,9] James R. Neilson,[7] Stephen D. Wilson,[2] Elif Ertekin,[3,4] Tyrel M. McQueen,[5] and Eric S. Toberer[1]
[1]*Colorado School of Mines, Golden, Colorado 80401, USA*
[2]*University of California Santa Barbara, Santa Barbara, California 93106, USA*
[3]*University of Illinois at Urbana-Champaign, Urbana, Illinois 61820, USA*
[4]*National Center for Supercomputing Applications, Urbana, Illinois 61801, USA*
[5]*Johns Hopkins University, Baltimore, Maryland 21218, USA*
[6]*Gdansk University of Technology, Gdansk 80-233, Poland*
[7]*Colorado State University, Fort Collins, Fort Collins, Colorado 80523, USA*
[8]*NIST Center for Neutron Research, Gaithersburg, Maryland 20878, USA*
[9]*University of Maryland, College Park, Maryland 20742, USA*







Object Categories

- Materials
  - Physical materials (very general)
- Processes
  - One or more materials in (“ingredients”), one material out (“output_material”)
- Ingredients
  - Enhancements to Materials describing how they’re used in a Process

Object Categories

- Materials
  - Physical materials (very general)
- Processes
  - One or more materials in (“ingredients”), one material out (“output_material”)
- Ingredients
  - Enhancements to Materials describing how they’re used in a Process

images from this food blog

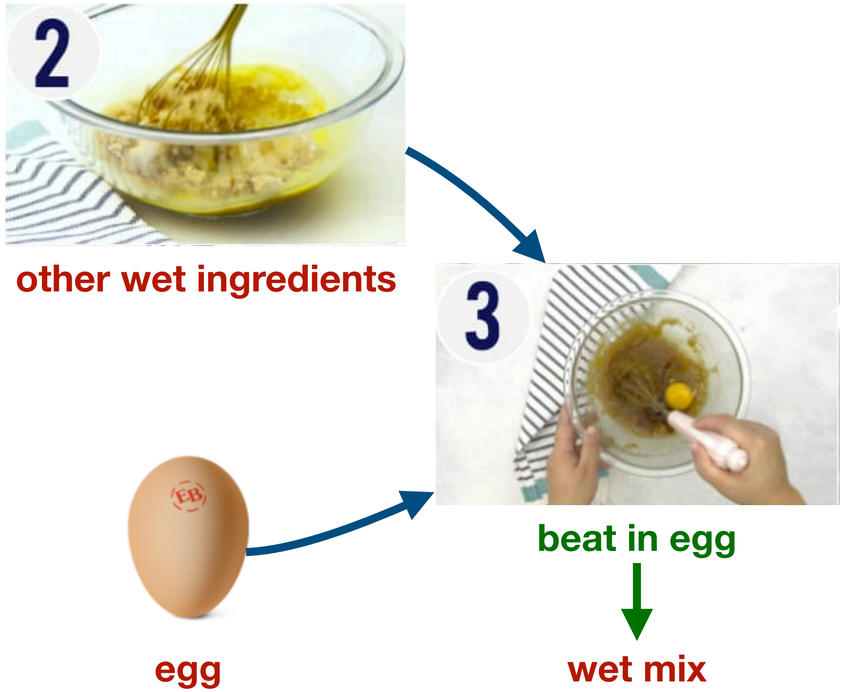
other wet ingredients

egg

Object Categories

- Materials
  - Physical materials (very general)
- Processes
  - One or more materials in (“ingredients”), one material out (“output_material”)
- Ingredients
  - Enhancements to Materials describing how they’re used in a Process

images from this food blog

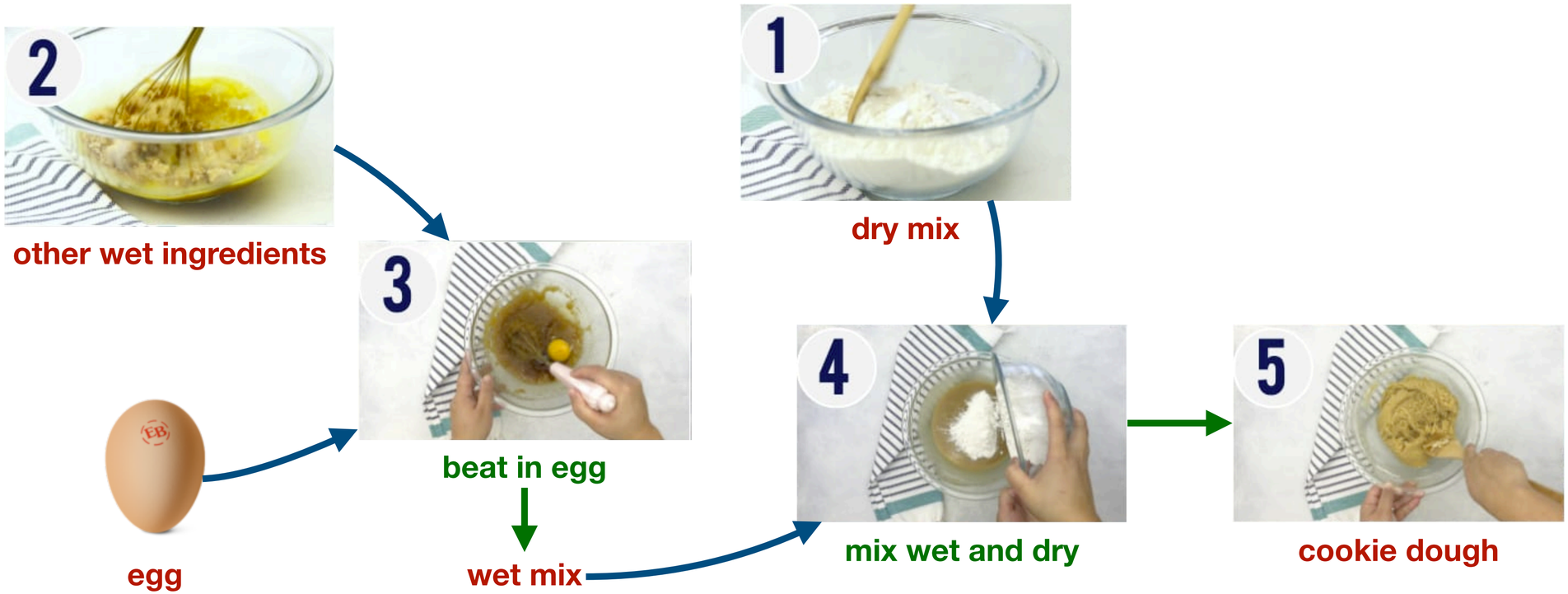


Object Categories

- Materials
  - Physical materials (very general)
- Processes
  - One or more materials in (“ingredients”), one material out (“output_material”)

- Ingredients
  - Enhancements to Materials describing how they’re used in a Process

images from this food blog



Object Categories

- Materials
  - Physical materials (very general)
- Processes
  - One or more materials in (“ingredients”), one material out (“output_material”)
- Ingredients
  - Enhancements to Materials describing how they’re used in a Process

images from this food blog

other wet ingredients

dry mix

chocolate chips

mix in chips

beat in egg

egg

wet mix

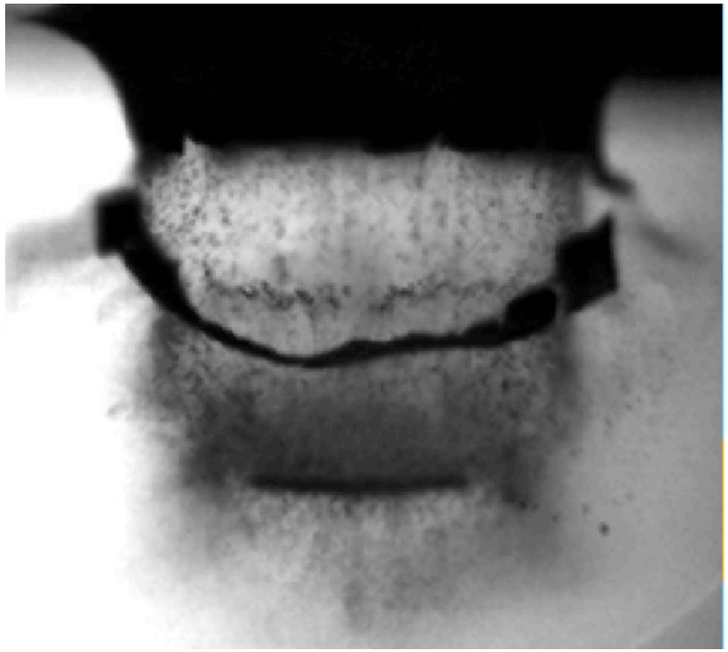
mix wet and dry

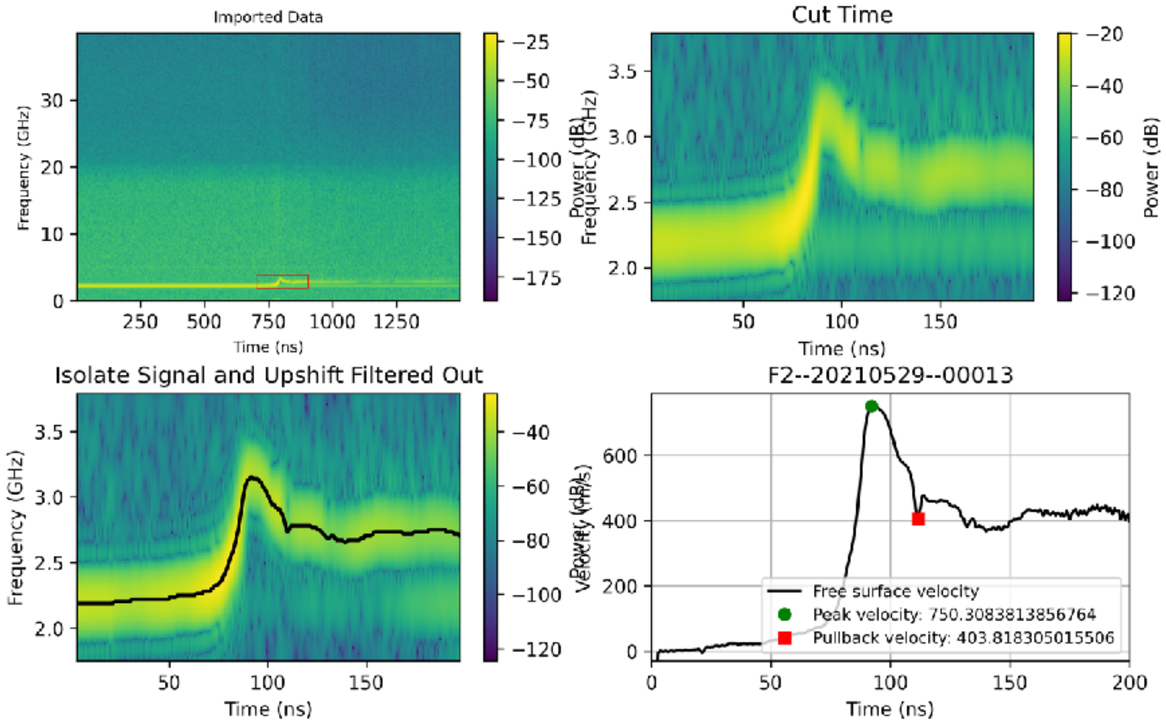
cookie dough

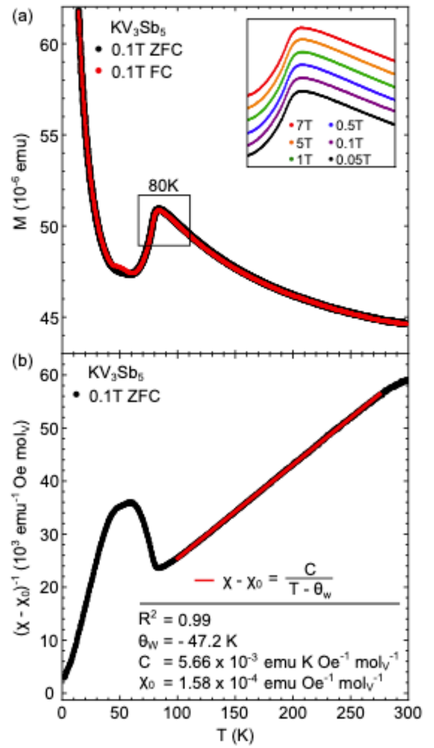
chocolate chip cookie dough

Object Categories

- Materials
  - Physical materials (very general)
- Processes
  - One or more materials in (“ingredients”), one material out (“output_material”)
- Ingredients
  - Enhancements to Materials describing how they’re used in a Process
- Measurements
  - Procedures for measuring properties of a Material







Attributes

- Properties
  - Something measured or calculated about a Material
  - Can be added for Materials and Measurements

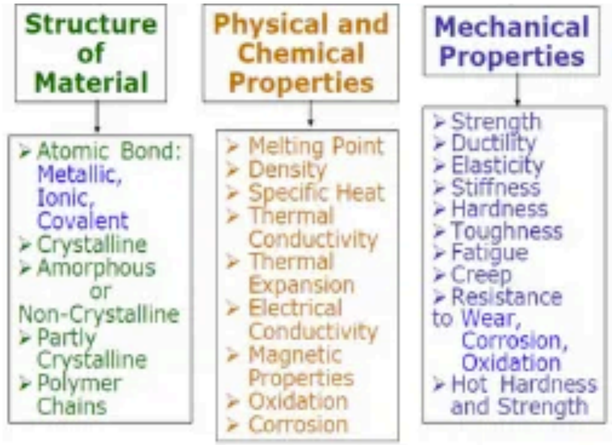


image sources:
1

Attributes

- Properties
  - Something measured or calculated about a Material
  - Can be added for Materials and Measurements
- Parameters
  - User-decided settings of a tool
  - Can be added for Processes and Measurements

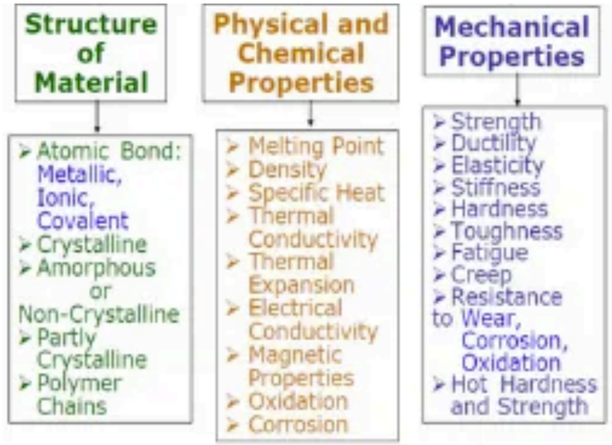


image sources:
1, 2

Attributes

- Properties
  - Something measured or calculated about a Material
  - Can be added for Materials and Measurements
- Parameters
  - User-decided settings of a tool
  - Can be added for Processes and Measurements
- Conditions
  - Passive aspects of environments or lab setups (types of equipment used)
  - Can be added for Processes and Measurements

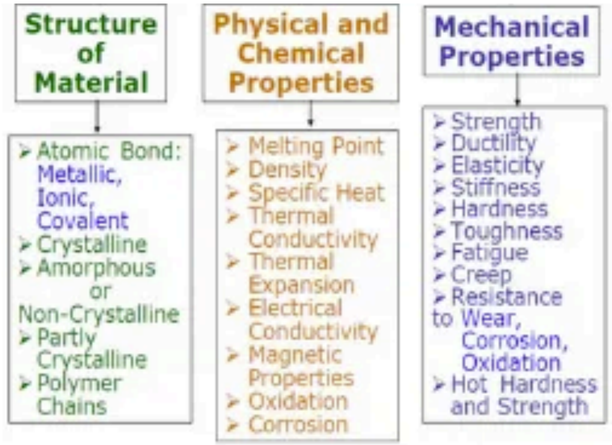


image sources:
1, 2, 3

Value Types

- Information contained in an Attribute
  - (Ingredient Specs/Runs also have some Values)

Type

- Real
- Integer
- Categorical
- Compositional
  - Empirical Formula
- Molecular Structure
  - SMILES or InChI

Distribution

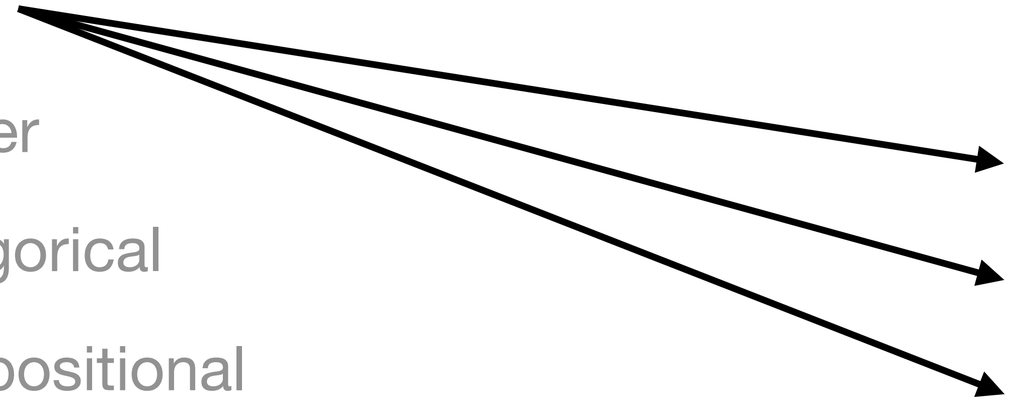
- Nominal
- Uniform
- Normal
- Discrete

Value Types

- Information contained in an Attribute
  - (Ingredient Specs/Runs also have some Values)

Type

- Real
- Integer
- Categorical
- Compositional
  - Empirical Formula
- Molecular Structure
  - SMILES or InChI



Distribution

- Nominal: single value + units
- Uniform: lower/upper bound + units
- Normal: mean, stddev + units
- Discrete

Value Types

- Information contained in an Attribute
  - (Ingredient Specs/Runs also have some Values)

Type

- Real
- Integer
- Categorical
- Compositional
  - Empirical Formula
- Molecular Structure
  - SMILES or InChI

Distribution

- Nominal: single value
- Uniform: lower/upper bound
- Normal
- Discrete

Value Types

- Information contained in an Attribute
  - (Ingredient Specs/Runs also have some Values)

Type

- Real
- Integer
- Categorical
- Compositional
  - Empirical Formula
- Molecular Structure
  - SMILES or InChI

Distribution

- Nominal: single (category) value
- Uniform
- Normal
- Discrete: values + probabilities

Value Types

- Information contained in an Attribute
  - (Ingredient Specs/Runs also have some Values)

Type

- Real
- Integer
- Categorical
- Compositional
  - Empirical Formula
- Molecular Structure
  - SMILES or InChI

Distribution

- Nominal: components + quantities
- Uniform
- Normal
- Discrete

Value Types

- Information contained in an Attribute
  - (Ingredient Specs/Runs also have some Values)

Type

- Real
- Integer
- Categorical
- Compositional
  - Empirical Formula
- Molecular Structure
  - SMILES or InChI

Distribution

- Nominal
- Uniform
- Normal
- Discrete

Templates, Specs, & Runs: High-level

- Templates
  - Define **the space of things** that can possibly occur
  - Validate and associate data
  - Attributes and Objects can both have templates

```python
from gemd.entity.template import ParameterTemplate
from gemd.entity.bounds import RealBounds
ATTR_TEMPL = {}

name = 'Oven Temperature'
ATTR_TEMPL[name] = ParameterTemplate(
    name=name,
    description='Temperature setting on the oven',
    bounds=RealBounds(100.,550.,'degF')
)
```

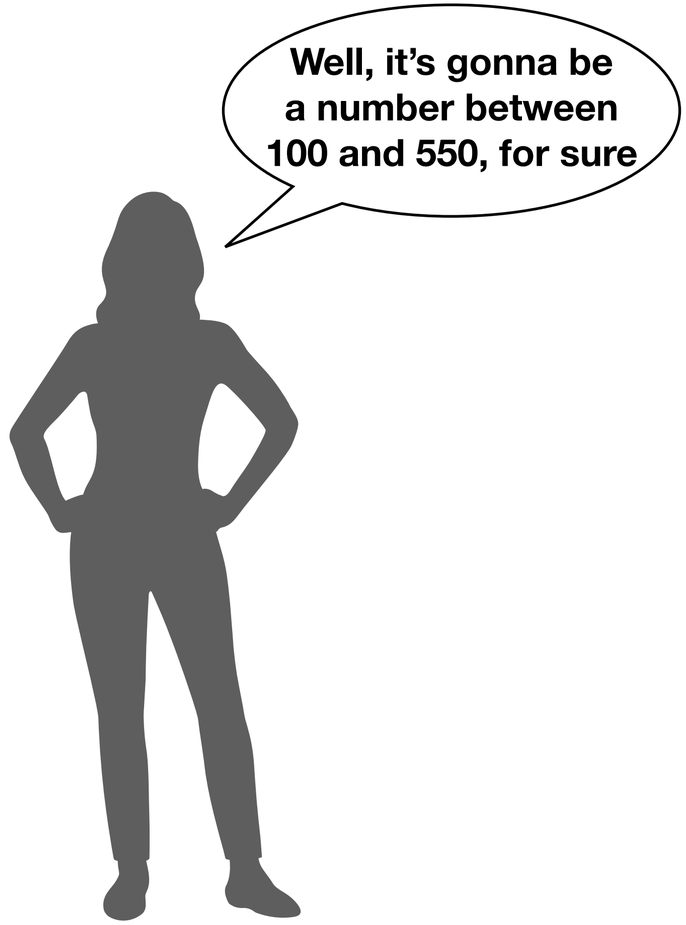


image source

Templates, Specs, & Runs: High-level

- Templates
  - Define **the space of things** that can possibly occur
  - Validate and associate data
  - Attributes and Objects can both have templates
- Specs
  - Represent **explicit/specific intent** to do something
  - Objects (only, all) have specs
  - Specs can reference corresponding Templates

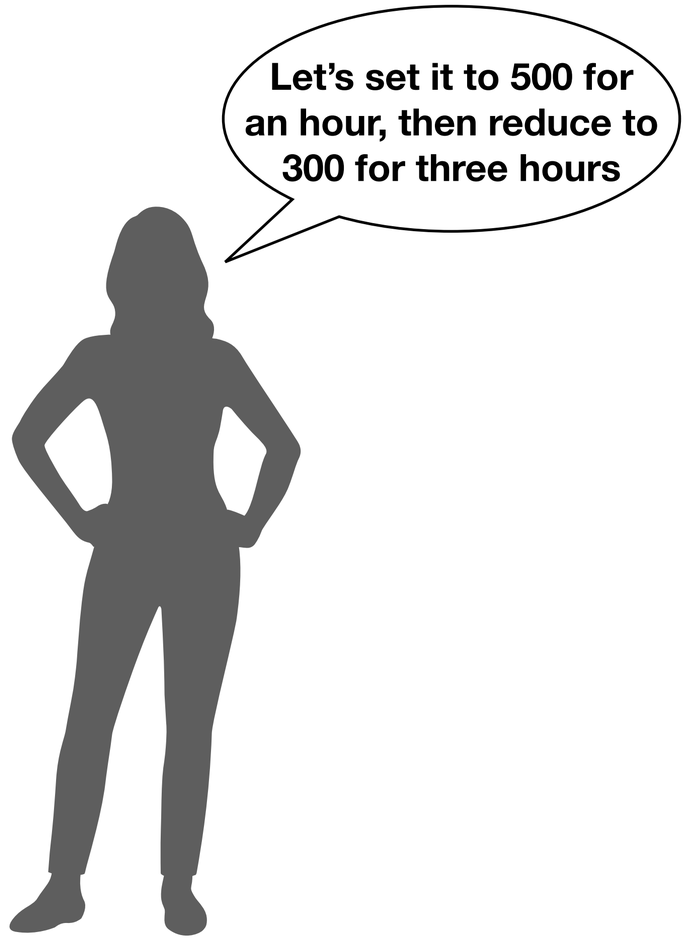


image source

Templates, Specs, & Runs: High-level

- Templates
  - Define **the space of things** that can possibly occur
  - Validate and associate data
  - Attributes and Objects can both have templates
- Specs
  - Represent **explicit/specific intent** to do something
  - Objects (only, all) can be in the Spec state
  - Specs can reference corresponding Templates
- Runs
  - The **actual realized result** of doing something
  - Objects (only, all) can be in the Run state
  - Usually reference Specs, but can differ from them

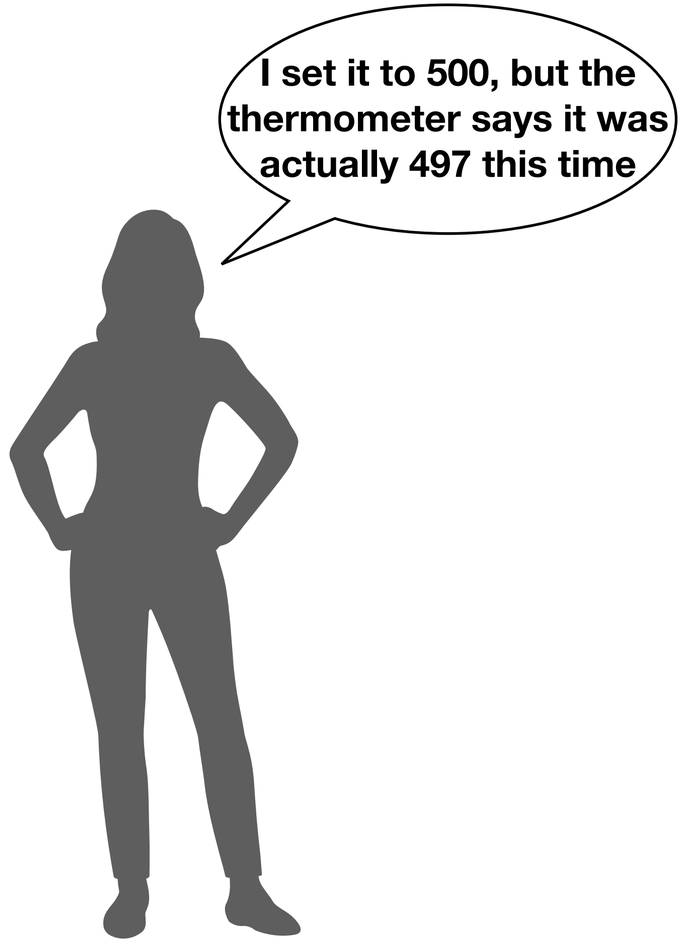


image source

Templates, Specs, & Runs

- Concepts
  - Templates exist before anything has even been done in a lab
    - Can literally be hardcoded

```
name = 'Glass ID'
OBJ_TEMPL[name] = MaterialTemplate(
    name=name,
    description='A piece of glass used in the Laser Shock Lab (used in creating Flyer Stacks)',
    properties = [ATTR_TEMPL['Glass Thickness'],
                  ATTR_TEMPL['Glass Length'],
                  ATTR_TEMPL['Glass Width'],
        ],
)

name = 'Purchasing Foil'
OBJ_TEMPL[name] = ProcessTemplate(
    name=name,
    description='Purchasing a foil from a manufacturer',
    parameters=[ATTR_TEMPL['Foil Supplier'],
                ATTR_TEMPL['Foil Part Number'],
        ],
    allowed_names=[],
)
```

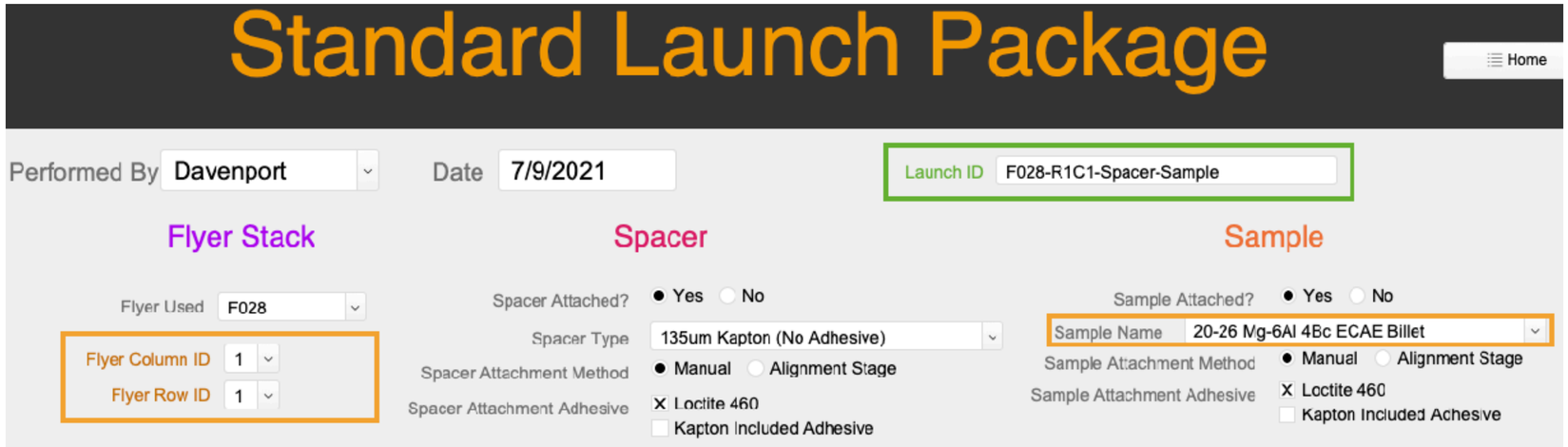
Some Object Templates

```
name = 'Annealing Time'
ATTR_TEMPL[name] = ParameterTemplate(
    name=name,
    description='Amount of time a Raw Material is annealed to produce a Sample',
    bounds=RealBounds(0,1e3,'hr'),
)

name = 'Polishing Pad'
ATTR_TEMPL[name] = ParameterTemplate(
    name=name,
    description='What was used to polish an impact sample intended for a Launch Package',
    bounds=CategoricalBounds(['Diamond','Silicon Carbide']),
)

name = 'Asymmetric Polish'
ATTR_TEMPL[name] = ParameterTemplate(
    name=name,
    description='Was the impact sample polished asymmetrically?',
    bounds=CategoricalBounds(['Yes','No']),
)

name = 'Polishing Side 2'
ATTR_TEMPL[name] = ParameterTemplate(
    name=name,
    description='Which was the other side of the sample that was polished?',
    bounds=CategoricalBounds(['Impact Side','PDV Side']),
)

name = 'Sample Location'
ATTR_TEMPL[name] = ParameterTemplate(
    name=name,
    description='The location of an impact sample',
    bounds=IntegerBounds(0,100),
)
```

Some Parameter Attribute Templates

Templates, Specs, & Runs

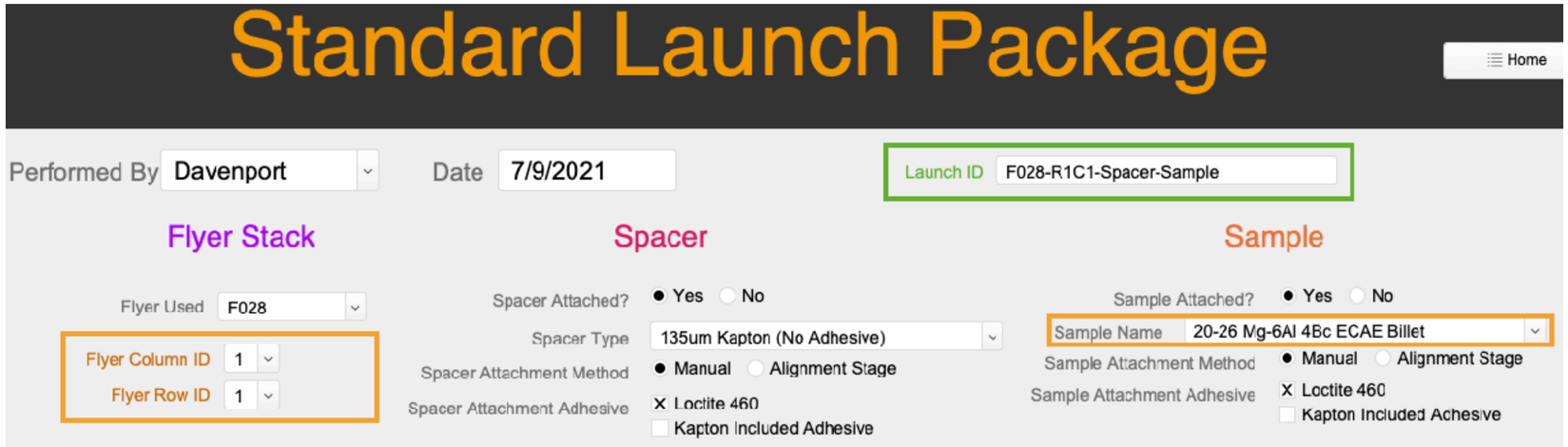
- Concepts
  - Templates exist before anything has even been done in a lab
    - Can literally be hardcoded
  - One Spec can be referenced by many Runs
  - In practice, Specs may need to be dynamically created



Templates, Specs, & Runs

- Concepts
  - Templates exist before anything has even been done in a lab
    - Can literally be hardcoded
  - One Spec can be referenced by many Runs
  - In practice, Specs may need to be dynamically created
  - In practice, you'll start creating an instance of a GEMD model by collecting all the Runs that exist and all the information about them

- A specific Flyer in a Flyer Stack was used
  - "flyer stack" material/ingredient spec/run
  - "choosing flyer" process w/ parameters
- A spacer was attached
  - "spacer type" material/ingredient spec
  - "attaching spacer" process spec/run w/ "method" parameter, "flyer"/"spacer"/"adhesive" material/ingredient specs/runs
- A sample was attached
  - you get the idea….



Further Details

Attribute Templates

```python
class AttributeTemplate(BaseEntity):
    """
    An attribute template, which can be a property, parameter, or condition template.
    """

    def __init__(self,
                 name,
                 *,
                 description=None,
                 bounds=None,
                 uids=None,
                 tags=None):
```

```python
class PropertyTemplate(AttributeTemplate):
    """A template for the property attribute.
    """
```

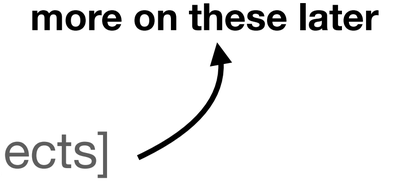
```python
class ParameterTemplate(AttributeTemplate):
    """A template for the parameter attribute.
    """
```

```python
class ConditionTemplate(AttributeTemplate):
    """A template for a condition attribute.
    """
```

Attribute Templates

```
class AttributeTemplate(BaseEntity):
    """
    An attribute template, which can be a property, parameter, or condition template.
    """

    def __init__(self,
                 name,
                 *,
                 description=None,
                 bounds=None,
                 uids=None,
                 tags=None):
```

- Name: Name for the object (string) [all Templates, Attributes, and Objects]
- Description: Long-form description of the object (string) [all Templates and Attributes]
- UIDs: List of (scope, UID) strings for UNIQUE descriptors of this object [all Templates and Objects]
- Tags: Additional information or values used for filtering or that doesn't fit elsewhere [all Templates and Objects]



Attribute Templates: Bounds

- Attribute templates are mostly just a name and a Bounds object constraining the Value their Attribute objects will describe
- The type of Bounds object in the AttributeTemplate depends on the type of Value object in the Attribute

- Real: lower/upper bound + default units
```
bounds=RealBounds(0.,100.,'mm'),
```
- Integer: lower/upper bound
```
bounds=IntegerBounds(0,20),
```
- Categorical: list of category names
```
bounds=CategoricalBounds(['Long pulse','Q switched'])
```
- Composition: list of component names
```
bounds=CompositionBounds(components=('Mg','Al','Zr','Ti','Cu','Ni','Be','Zn','Ca'))
```
- Molecular structure

Attributes

```python
class BaseAttribute(DictSerializable):
    """
    Base class for all attributes, which include property, condition, parameter, and metadata.
    """

    def __init__(self,
                 name: str,
                 *,
                 template: Union[AttributeTemplate, LinkByUID, None] = None,
                 origin: Union[Origin, str] = Origin.UNKNOWN,
                 value: BaseValue = None,
                 notes: str = None,
                 file_links: Optional[Union[Iterable[FileLink], FileLink]] = None):
```

```python
class Property(BaseAttribute):
    """
    Property of a material, measured in a MeasurementRun or specified in a MaterialSpec.
    """
```

```python
class Parameter(BaseAttribute):
    """
    Parameter of a process or measurement.

    Parameters are the non-environmental variables (typically specified and controlled) that may
    affect a process or measurement: e.g. oven dial temperature for a kiln firing, magnification
    for a measurement taken with an electron microscope.
    """
```

```python
class Condition(BaseAttribute):
    """
    Condition of a property, process, or measurement.

    Conditions are environmental variables (typically measured) that may affect a process
    or measurement: e.g., temperature, pressure.
    """
```

Attributes

```python
class BaseAttribute(DictSerializable):
    """
    Base class for all attributes, which include property, condition, parameter, and metadata.
    """

    def __init__(self,
                 name: str,
                 *,
                 template: Union[AttributeTemplate, LinkByUID, None] = None,
                 origin: Union[Origin, str] = Origin.UNKNOWN,
                 value: BaseValue = None,
                 notes: str = None,
                 file_links: Optional[Union[Iterable[FileLink], FileLink]] = None):
```

- Origin: “measured”, “predicted”, “summary”, “specified”, “computed”, “unknown” [all Attributes]
- Notes: Additional longform comments/notes (string) [all Attributes and Objects]
- FileLink: Small class that takes a file name and URL [all Attributes and Objects]

Attributes

```python
class PropertyAndConditions(DictSerializable):
    """
    A property and the conditions under which that property was determined.

    This attribute is only relevant for material specs.
    """

    typ = "property_and_conditions"

    def __init__(self,
                 property: Property = None,
                 conditions: Iterable[Condition] = None):
```

Object Templates

```python
class MaterialTemplate(BaseTemplate, HasPropertyTemplates):
    """
    A material template.

    Material templates are collections of property templates that constrain the values of
    a material's property attributes, and provide a common structure for describing similar
    materials.
    """

    typ = "material_template"

    def __init__(self,
                 name,
                 *,
                 description=None,
                 properties=None,
                 uids=None,
                 tags=None):
```

Object Templates

```python
class ProcessTemplate(BaseTemplate, HasConditionTemplates, HasParameterTemplates):
    """
    A process template.

    Process templates are collections of condition and parameter templates that constrain the
    values of a measurement's condition and parameter attributes, and provide a common structure
    for describing similar measurements.
    """

    typ = "process_template"

    def __init__(self,
                 name,
                 *,
                 description=None,
                 conditions=None,
                 parameters=None,
                 allowed_names=None,
                 allowed_labels=None,
                 uids=None,
                 tags=None):
```

- allowed_names/labels: unique to ProcessTemplates, constrain permitted ingredients

Object Templates

```python
class MeasurementTemplate(BaseTemplate,
                          HasPropertyTemplates, HasConditionTemplates, HasParameterTemplates):
    """
    A measurement template.

    Measurement templates are collections of condition, parameter and property templates that
    constrain the values of a measurement's condition, parameter and property attributes, and
    provide a common structure for describing similar measurements.
    """

    typ = "measurement_template"

    def __init__(self,
                 name,
                 *,
                 description=None,
                 properties=None,
                 conditions=None,
                 parameters=None,
                 uids=None,
                 tags=None):
```

(Object) Specs & Runs: Materials

```python
class MaterialSpec(BaseObject, HasTemplate, HasProcess, HasProperties):
    """
    A material specification.

    This includes a link to the originating process and specified properties with conditions.
    """

    typ = "material_spec"

    def __init__(self,
                 name: str,
                 *,
                 template: Optional[Union[MaterialTemplate, LinkByUID]] = None,
                 process: Union[ProcessSpec, LinkByUID] = None,
                 properties: Iterable[PropertyAndConditions] = None,
                 uids: Mapping[str, str] = None,
                 tags: Iterable[str] = None,
                 notes: str = None,
                 file_links: Optional[Union[Iterable[FileLink], FileLink]] = None):
```

- Run -> Spec -> Template

```python
class MaterialRun(BaseObject, HasSpec, HasProcess):
    """
    A material run.

    This includes a link to the originating process and soft links to measurements.
    """

    typ = "material_run"

    skip = {"_measurements"}

    def __init__(self,
                 name: str,
                 *,
                 spec: Union[MaterialSpec, LinkByUID] = None,
                 process: Union[ProcessRun, LinkByUID] = None,
                 sample_type: Union[SampleType, str] = "unknown",
                 uids: Mapping[str, str] = None,
                 tags: Iterable[str] = None,
                 notes: str = None,
                 file_links: Optional[Union[Iterable[FileLink], FileLink]] = None):
```

(Object) Specs & Runs: Materials

```python
class MaterialSpec(BaseObject, HasTemplate, HasProcess, HasProperties):
    """
    A material specification.

    This includes a link to the originating process and specified properties with conditions.
    """

    typ = "material_spec"

    def __init__(self,
                 name: str,
                 *,
                 template: Optional[Union[MaterialTemplate, LinkByUID]] = None,
                 process: Union[ProcessSpec, LinkByUID] = None,
                 properties: Iterable[PropertyAndConditions] = None,
                 uids: Mapping[str, str] = None,
                 tags: Iterable[str] = None,
                 notes: str = None,
                 file_links: Optional[Union[Iterable[FileLink], FileLink]] = None):
```

- Run -> Spec -> Template
- LinkByUID: used in serialization. Any pointers to objects can be replaced with LinkByUID objects
- sample_type: “experimental”, “production”, “virtual”, “unknown”

```python
class MaterialRun(BaseObject, HasSpec, HasProcess):
    """
    A material run.

    This includes a link to the originating process and soft links to measurements.
    """

    typ = "material_run"

    skip = {"_measurements"}

    def __init__(self,
                 name: str,
                 *,
                 spec: Union[MaterialSpec, LinkByUID] = None,
                 process: Union[ProcessRun, LinkByUID] = None,
                 sample_type: Union[SampleType, str] = "unknown",
                 uids: Mapping[str, str] = None,
                 tags: Iterable[str] = None,
                 notes: str = None,
                 file_links: Optional[Union[Iterable[FileLink], FileLink]] = None):
```

(Object) Specs & Runs: Materials

```python
class MaterialSpec(BaseObject, HasTemplate, HasProcess, HasProperties):
    """
    A material specification.

    This includes a link to the originating process and specified properties with conditions.
    """

    typ = "material_spec"

    def __init__(self,
                 name: str,
                 *,
                 template: Optional[Union[MaterialTemplate, LinkByUID]] = None,
                 process: Union[ProcessSpec, LinkByUID] = None,
                 properties: Iterable[PropertyAndConditions] = None,
                 uids: Mapping[str, str] = None,
                 tags: Iterable[str] = None,
                 notes: str = None,
                 file_links: Optional[Union[Iterable[FileLink], FileLink]] = None):
```

```python
class MaterialRun(BaseObject, HasSpec, HasProcess):
    """
    A material run.

    This includes a link to the originating process and soft links to measurements.
    """

    typ = "material_run"

    skip = {"_measurements"}

    def __init__(self,
                 name: str,
                 *,
                 spec: Union[MaterialSpec, LinkByUID] = None,
                 process: Union[ProcessRun, LinkByUID] = None,
                 sample_type: Union[SampleType, str] = "unknown",
                 uids: Mapping[str, str] = None,
                 tags: Iterable[str] = None,
                 notes: str = None,
                 file_links: Optional[Union[Iterable[FileLink], FileLink]] = None):
```

Bidirectional Links

- Setting MaterialSpec/Run “process” makes the material that ProcessSpec/Run’s “output_material”
- Setting a MeasurementRun’s “material” adds it to the MaterialRun’s list of “measurements”

(Object) Specs & Runs: Processes

```python
class ProcessSpec(BaseObject, HasTemplate, HasParameters, HasConditions):
    """
    A process specification.

    Processes transform zero or more input materials into exactly one output material.
    This includes links to the parameters and conditions under which the process is expected
    to be performed, as well as soft links to the output material and the input ingredients.
    """

    typ = "process_spec"

    skip = {"_output_material", "_ingredients"}

    def __init__(self,
                 name: str,
                 *,
                 template: Optional[Union[ProcessTemplate, LinkByUID]] = None,
                 conditions: Iterable[Condition] = None,
                 parameters: Iterable[Parameter] = None,
                 uids: Mapping[str, str] = None,
                 tags: Iterable[str] = None,
                 notes: str = None,
                 file_links: Optional[Union[Iterable[FileLink], FileLink]] = None):
```

```python
class ProcessRun(BaseObject, HasSpec, HasConditions, HasParameters, HasSource):
    """
    A process run.

    Processes transform zero or more input materials into exactly one output material.
    This includes links to conditions and parameters under which the process was performed,
    as well as soft links to the output material and each of the input ingredients.
    """

    typ = "process_run"

    skip = {"_output_material", "_ingredients"}

    def __init__(self,
                 name: str,
                 *,
                 spec: Union[ProcessSpec, LinkByUID] = None,
                 conditions: Iterable[Condition] = None,
                 parameters: Iterable[Parameter] = None,
                 uids: Mapping[str, str] = None,
                 tags: Iterable[str] = None,
                 notes: str = None,
                 file_links: Optional[Union[Iterable[FileLink], FileLink]] = None,
                 source: PerformedSource = None):
```

- PerformedSource: small class that takes “performed_by” and “performed_date” strings

(Object) Specs & Runs: Processes

```python
class ProcessSpec(BaseObject, HasTemplate, HasParameters, HasConditions):
    """
    A process specification.

    Processes transform zero or more input materials into exactly one output material.
    This includes links to the parameters and conditions under which the process is expected
    to be performed, as well as soft links to the output material and the input ingredients.
    """

    typ = "process_spec"

    skip = {"_output_material", "_ingredients"}

    def __init__(self,
                 name: str,
                 *,
                 template: Optional[Union[ProcessTemplate, LinkByUID]] = None,
                 conditions: Iterable[Condition] = None,
                 parameters: Iterable[Parameter] = None,
                 uids: Mapping[str, str] = None,
                 tags: Iterable[str] = None,
                 notes: str = None,
                 file_links: Optional[Union[Iterable[FileLink], FileLink]] = None):
```

```python
class ProcessRun(BaseObject, HasSpec, HasConditions, HasParameters, HasSource):
    """
    A process run.

    Processes transform zero or more input materials into exactly one output material.
    This includes links to conditions and parameters under which the process was performed,
    as well as soft links to the output material and each of the input ingredients.
    """

    typ = "process_run"

    skip = {"_output_material", "_ingredients"}

    def __init__(self,
                 name: str,
                 *,
                 spec: Union[ProcessSpec, LinkByUID] = None,
                 conditions: Iterable[Condition] = None,
                 parameters: Iterable[Parameter] = None,
                 uids: Mapping[str, str] = None,
                 tags: Iterable[str] = None,
                 notes: str = None,
                 file_links: Optional[Union[Iterable[FileLink], FileLink]] = None,
                 source: PerformedSource = None):
```

Bidirectional Links

- Setting MaterialSpec/Run's "process" makes the material that ProcessSpec/Run's "output_material"
- Setting IngredientSpec/Run's "process" adds it to the ProcessSpec/Run's list of "ingredients"

(Object) Specs & Runs: Ingredients

```python
class IngredientSpec(BaseObject, HasQuantities, HasTemplate, HasMaterial, HasProcess):
    """
    An ingredient specification.

    Ingredients annotate a material with information about its usage in a process.
    """

    typ = "ingredient_spec"

    def __init__(self,
                 name: str,
                 *,
                 material: Union[MaterialSpec, LinkByUID] = None,
                 process: Union[ProcessSpec, LinkByUID] = None,
                 labels: Iterable[str] = None,
                 mass_fraction: ContinuousValue = None,
                 volume_fraction: ContinuousValue = None,
                 number_fraction: ContinuousValue = None,
                 absolute_quantity: ContinuousValue = None,
                 uids: Mapping[str, str] = None,
                 tags: Iterable[str] = None,
                 notes: str = None,
                 file_links: Optional[Union[Iterable[FileLink], FileLink]] = None):
```

- “process” writes a bidirectional link
- labels: strings, checked again “allowed_labels”
- ContinuousValue fields describe amounts of Materials used in the Process

```python
class IngredientRun(BaseObject, HasQuantities, HasSpec, HasMaterial, HasProcess):
    """
    An ingredient run.

    Ingredients annotate a material with information about its usage in a process.
    """

    typ = "ingredient_run"

    def __init__(self,
                 *,
                 material: Union[MaterialRun, LinkByUID] = None,
                 process: Union[ProcessRun, LinkByUID] = None,
                 mass_fraction: ContinuousValue = None,
                 volume_fraction: ContinuousValue = None,
                 number_fraction: ContinuousValue = None,
                 absolute_quantity: ContinuousValue = None,
                 spec: Union[IngredientSpec, LinkByUID] = None,
                 uids: Mapping[str, str] = None,
                 tags: Iterable[str] = None,
                 notes: str = None,
                 file_links: Optional[Union[Iterable[FileLink], FileLink]] = None):
```

(Object) Specs & Runs: Measurements

```python
class MeasurementSpec(BaseObject, HasTemplate, HasParameters, HasConditions):
    """
    A measurement specification.

    This includes links to the conditions and parameters under which the measurement is
    expected to be performed.
    """

    typ = "measurement_spec"

    def __init__(self,
                 name: str,
                 *,
                 template: Optional[Union[MeasurementTemplate, LinkByUID]] = None,
                 conditions: Iterable[Condition] = None,
                 parameters: Iterable[Parameter] = None,
                 uids: Mapping[str, str] = None,
                 tags: Iterable[str] = None,
                 notes: str = None,
                 file_links: Optional[Union[Iterable[FileLink], FileLink]] = None):
```

```python
class MeasurementRun(BaseObject, HasMaterial, HasSpec, HasConditions, HasProperties,
                     HasParameters, HasSource):
    """
    A measurement run.

    This contains a link to the material the measurement is performed on, as well as links to
    any properties, conditions, and parameters.
    """

    typ = "measurement_run"

    def __init__(self,
                 name: str,
                 *,
                 spec: Union[MeasurementSpec, LinkByUID] = None,
                 material: Union[MaterialRun, LinkByUID] = None,
                 properties: Iterable[Property] = None,
                 conditions: Iterable[Condition] = None,
                 parameters: Iterable[Parameter] = None,
                 uids: Mapping[str, str] = None,
                 tags: Iterable[str] = None,
                 notes: str = None,
                 file_links: Optional[Union[Iterable[FileLink], FileLink]] = None,
                 source: PerformedSource = None):
```

- “material” writes a bidirectional link

GEMD (again)

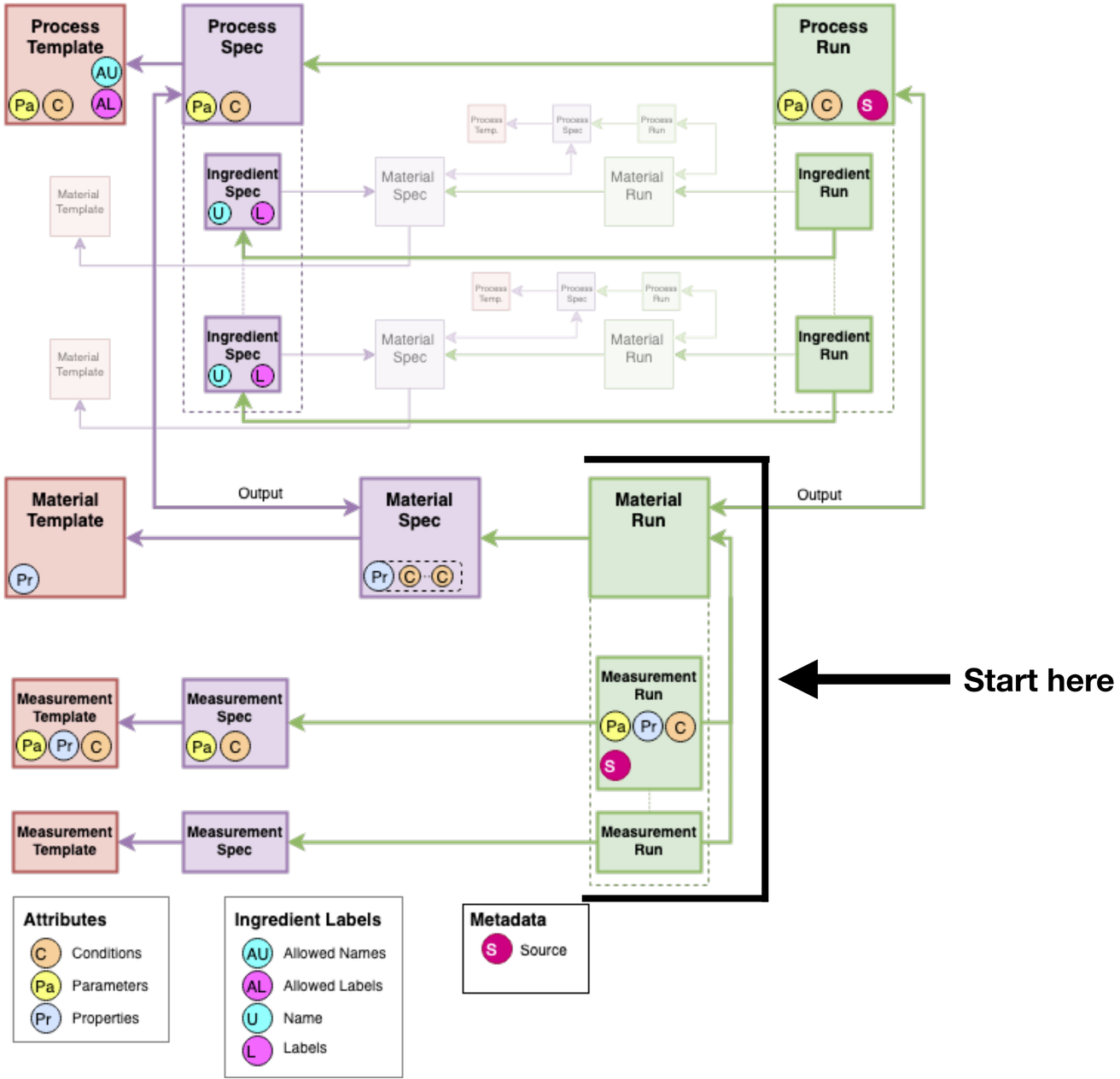
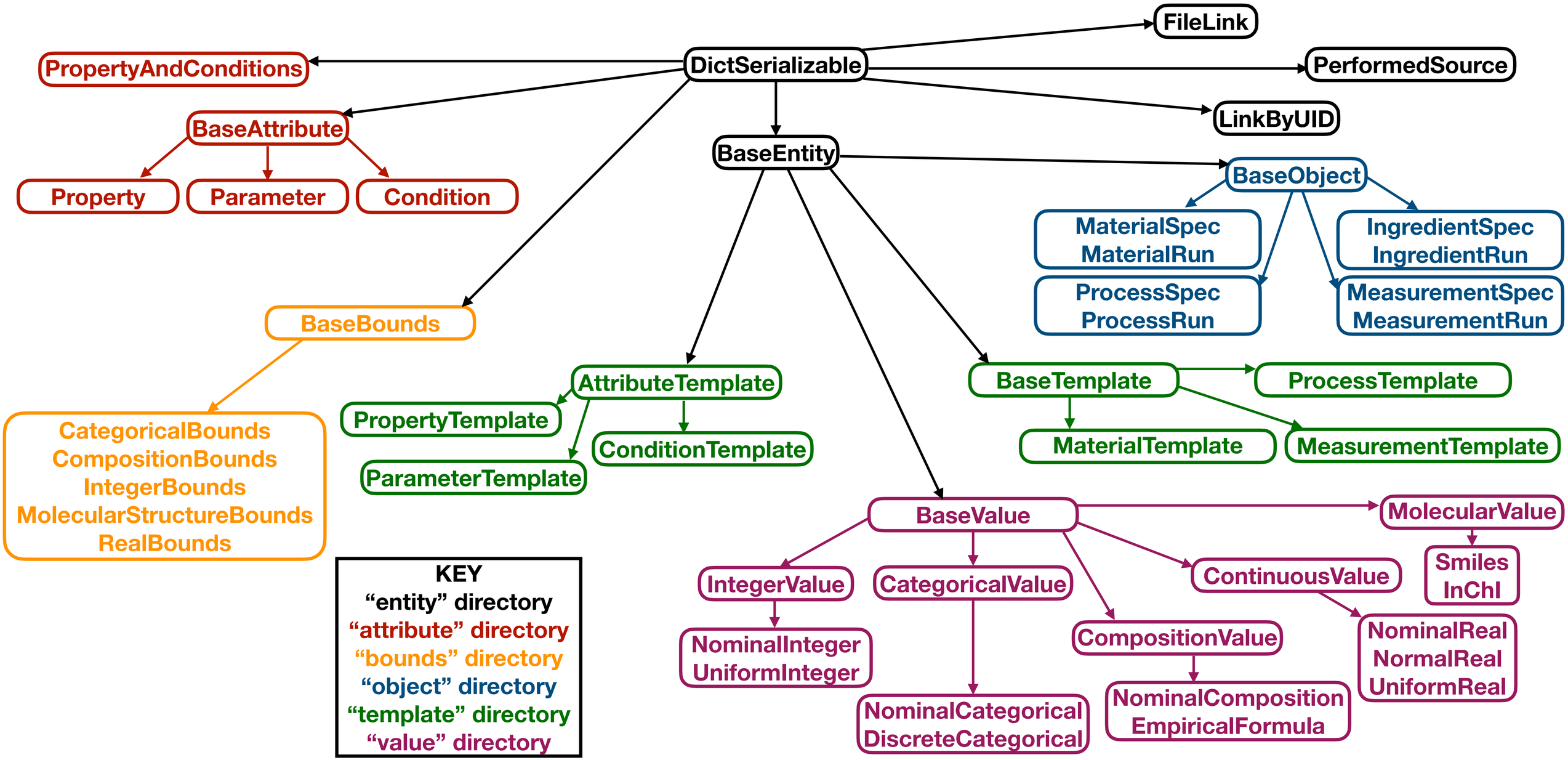


image from
GEMD documentation

Practical Concerns

(some of the) inheritance in `gemd-python`



DictSerializable

- Just about everything inherits from DictSerializable
- DictSerializable is anything that can be represented as a Python dictionary and serialized to a string
- Can call “as_dict” on any object to get its representation as a dictionary
- You’ll see everything in that dictionary EXCEPT what’s in “skip” for the object type
  - Prevents infinitely looping through bidirectional links

```
129    name = 'Sample Material Type'
130    ATTR_TEMPL[name] = PropertyTemplate(
131        name=name,
132        description='Possible values in the "Material" menu buttons in the "Sample" layout',
133        bounds=CategoricalBounds(['Metal','Ceramic','Polymer','BMG','HEA','Composite'])
134    )
```


```
{"bounds": {"categories": ["BMG", "Ceramic", "Composite", "HEA", "Metal", "Polymer"],
"type": "categorical_bounds"}, "description": "Possible values in the \"Material\" menu
buttons in the \"Sample\" layout", "name": "Sample Material Type", "tags": [], "type":
"property_template", "uids": {"auto": "b6abf464-7dd6-4f0e-b813-9ca5dfb2f419"}}
```

Serialization

- An instance of the GEMD model is a big graph of objects
  - Some of the edges of the graph point in two directions, but can only be written in one
- Serializing an instance of the model (a collection of objects) requires a few steps:
  - Assign UIDs for every object
  - Traverse and flatten the graph into a list, replacing links to objects with LinkByUID objects
  - Sort the list into a special “writeable” order
  - Write out the list of objects
    - top-level objects go in “object”
    - other objects go in “context”’
- Use GEMD’s “GEMDJson” class to serialize (“dump”) and deserialize (“load”) groups of objects representing instances of models

```python
def dumps(self, obj, **kwargs):
    """
    Serialize a gemd object, or container of them, into a json-formatting string.

    Parameters
    ----------
    obj: DictSerializable or List[DictSerializable]
        The object(s) to serialize to a string.
    **kwargs: keyword args, optional
        Optional keyword arguments to pass to `json.dumps()`.

    Returns
    -------
    str
        A string version of the serialized objects.

    """
    # create a top level list of [flattened_objects, link-i-fied return value]
    res = {"object": obj}

    additional = flatten(res, self.scope)
    res = substitute_links(res)
    res["context"] = additional
    return json_builtin.dumps(res, cls=GEMDEncoder, sort_keys=True, **kwargs)
```

the code

the output


```
{
  "context": [
    {
      "conditions": [],
      "file_links": [],
      "name": "producing process",
      "notes": null,
      "parameters": [],
      "tags": [],
      "template": null,
      "type": "process_spec",
      "uids": {
        "auto": "a103b759-b3e9-472e-8ec1-c69ee5d1981a"
      }
    },
    {
      "file_links": [],
      "name": "Produced material",
      "notes": null,
      "process": {
        "id": "a103b759-b3e9-472e-8ec1-c69ee5d1981a",
        "scope": "auto",
        "type": "link_by_uid"
      },
      "properties": [],
      "tags": [],
      "template": null,
      "type": "material_spec",
      "uids": {
        "auto": "ad2c31ab-e8c0-40f1-a1b6-c5b5950026cd"
      }
    }
  ],
  "object": {
    "id": "ad2c31ab-e8c0-40f1-a1b6-c5b5950026cd",
    "scope": "auto",
    "type": "link_by_uid"
  }
}
```

`make_instance` and `recursive_foreach`

- These are two super-useful utility functions
- "`make_instance`" takes a Spec object and returns a Run object created from the Spec, with all of its linked Spec objects also recursively replaced with corresponding Run objects
  - Invaluable when working with dynamically-created Specs

```
def make_instance(base_spec):
    """
    Create a set of Run objects that mimic the connectivity of the passed Spec object.

    Paramters
    _________
    base_spec: BaseObject
        A spec instance that may point to other specs.

    Returns
    _______
    BaseObject
        The run instance that is created, and may point to other runs.

    """
```

`make_instance` **and** `recursive_foreach`

- These are two super-useful utility functions
- "`make_instance`" takes a Spec object and returns a Run object created from the Spec, with all of its linked Spec objects also recursively replaced with corresponding Run objects
  - Invaluable when working with dynamically-created Specs
- "`recursive_foreach`" will run a function on every BaseEntity object found recursively from a starting point
- there are some other useful functions in `gemd.util.impl.py`

```
def recursive_foreach(obj: Union[Iterable, BaseEntity, DictSerializable],
                      func: Callable[[BaseEntity], None],
                      *,
                      apply_first=False):
    """
    Apply a function recursively to each BaseEntity object.

    Only objects of type BaseEntity will have the function applied, but the recursion will walk
    through all objects.  For example, BaseEntity -> list -> BaseEntity will have func applied
    to both base entities.

    Parameters
    ----------
    obj: Union[Iterable, Mapping, BaseEntity, DictSerializable]
        target of the operation
    func: Callable[[BaseEntity], None]
        to apply to each contained BaseEntity
    apply_first: bool
        whether to apply the func before applying it to members (default: false)

    Returns
    -------
    None

    """
```

Further Resources

- **Official GEMD Documentation:** https://citrineinformatics.github.io/gemd-docs/
- **Addendum for gemd-python:** https://citrineinformatics.github.io/gemd-python/depth/serialization.html
  - (minimal, but helpful for serialization)
- **gemd-python GitHub repo:** https://github.com/CitrineInformatics/gemd-python
  - cake example: https://github.com/CitrineInformatics/gemd-python/blob/main/gemd/demo/cake.py
- **OpenMSIPython:** https://github.com/openmsi/openmsipython/tree/main/openmsipython/data_models
  - **GEMDTemplateStore:** https://github.com/openmsi/openmsipython/blob/main/openmsipython/data_models/gemd_template_store.py
  - **GEMDSpecStore:** https://github.com/openmsi/openmsipython/blob/main/openmsipython/data_models/gemd_spec_store.py
- **And me!** >:3c (margaret.eminizer@gmail.com)