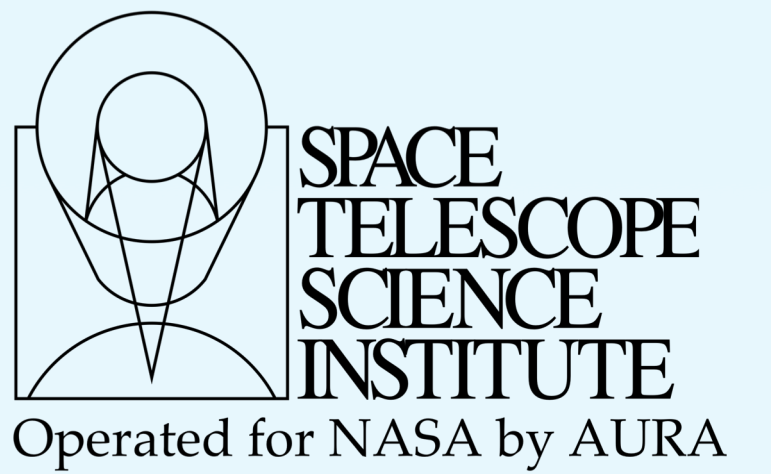




JWST Association Generator

Jonathan Eisenhamer (STScI)

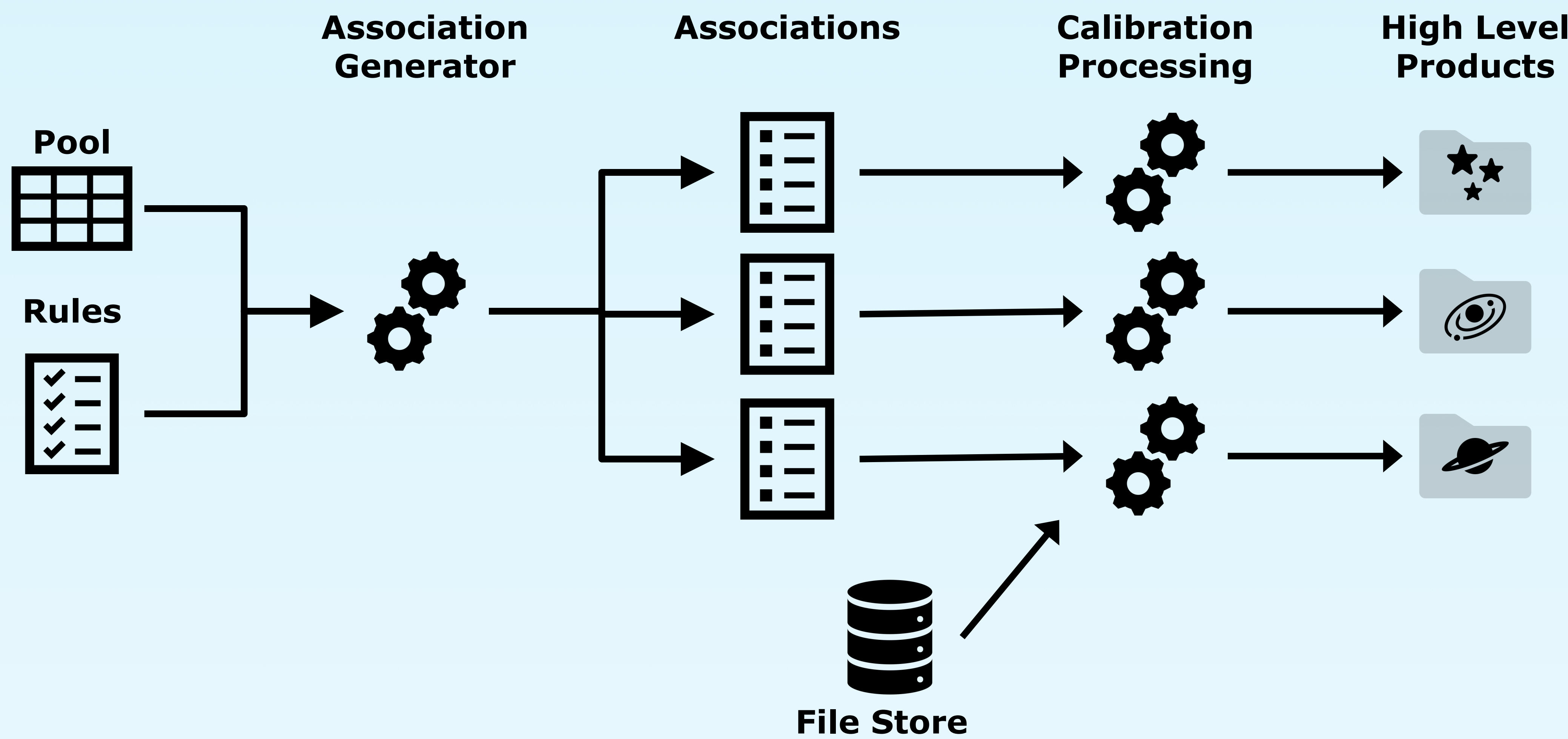


Overview

An Association is a list of items that are in some way related. The Association Generator, through a set of Rules, filters a larger list of items, called a Pool, producing a set of Associations.

For JWST, the Pool is the list of exposures produced by the observatory. The resulting associations are groups of exposures which will form higher level data products. The type of association created determines what type of downstream processing will be performed.

Workflow



API

The default command to run a pool against the Stage 2 and Stage 3 JWST rules is:

```
$ asn_generate pool.csv
```

Command to use a user-specified set of rules:

```
$ asn_generate pool.csv -r my_rules.py
```

From Python, the full command-line interface is accessed using the Main class:

```
from jwst.associations.main import Main as asn_generate
```

```
result = asn_generate(['pool.csv'])
```

```
result.associations    # Associations generated
result.orphaned        # Table of rows not placed in any associations
```

The typical full programmatic interface would be as follows:

```
from jwst.associations import AssociationRegistry, AssociationPool, generate
```

```
pool = AssociationPool.read('pool.csv')
rules = AssociationRegistry()           # Use the default JWST rules
associations = generate(pool, rules)    # Returns the list of associations
```

Associations

Associations created by the default JWST rules are simply Python dicts. The associations are serialized to files using JSON. YAML is also fully supported. Each association consists of metadata and a product list. Each product represents a single calibration process. Each product consists of a list of input exposure files and a suggested base name for the output.

An example Stage 3 association is:

```
{
  "asn_type": "coron3",
  "asn_rule": "candidate_Asn_Coron",
  "version_id": null,
  "code_version": "0.12.0a.dev181",
  "degraded_status": "No known degraded exposures in association.",
  "program": "10005",
  "asn_id": "c1007",
  "target": "t001",
  "asn_pool": "jw10005_20181020T033546_pool",
  "products": [
    {
      "name": "jw10005-c1007_t001_nircam_f430m-maskrnd-sub320a430r",
      "members": [
        {
          "expname": "jw10005009001_02102_00001_nrcalong_calints.fits",
          "exptype": "psf",
          "exposerr": "null",
          "asn_candidate": "('c1007', 'coronagraphy)"
        },
        {
          "expname": "jw10005006001_02102_00001_nrcalong_calints.fits",
          "exptype": "science",
          "exposerr": "null",
          "asn_candidate": "('c1007', 'coronagraphy)"
        }
      ]
    }
  ]
}
```

Rules

Rules are Python classes that define both the association structure and what members from the Pool will constitute an association.

JWST rules fall into two groups: Stage 2 and Stage 3 associations, and are used as input to the corresponding State 2 and Stage 3 calibration pipelines .

A example Stage 3 rule is:

```
@RegistryMarker.rule
class Asn_SpectralSource(AsnMixin_Spectrum):
    """Slit-like, multi-object spectrographic modes"""

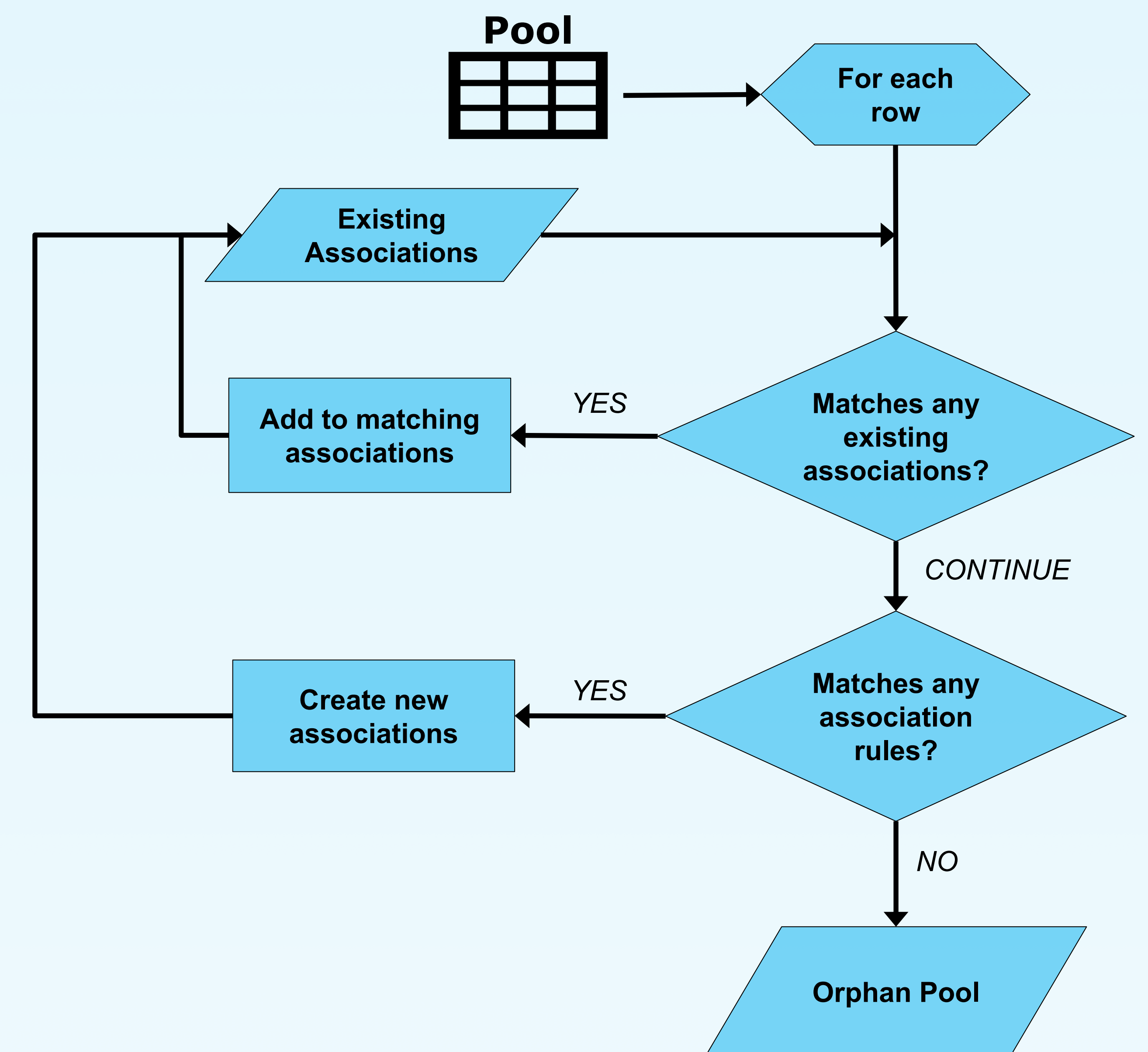
    def __init__(self, *args, **kwargs):

        # Setup for checking.
        self.constraints = Constraint([
            Constraint(
                [Constraint_TSO()],
                reduce=Constraint.notany
            ),
            Constraint_Optical_Path(),
            Constraint_Target(),
            Constraint(
                [
                    DMSAttrConstraint(
                        name='exp_type',
                        sources=['exp_type'],
                        value=(
                            'nrc_wfss'
                            'nirs_autoflat'
                            'nirs_autowave'
                            'nirs_fixedslit'
                        ),
                        force_unique=False
                    ),
                    Constraint_MSA()
                ],
                reduce=Constraint.any
            )
        ])

        # Check and continue initialization.
        super(Asn_SpectralSource, self).__init__(*args, **kwargs)

    @property
    def dms_product_name(self):
        return dms_product_name_sources(self)
```

Generator Algorithm (simplified)



Pools

Pools are tables containing the attributes of each exposure needed to assign exposures to associations. Any exposure can be part of zero to many different associations. Pools can be any format which astropy tables can read. An example of a partial pool is:

filename	obs_num	detector	filter	pupil
jw10002001001_01101_00006_nrcb2_uncal.fits	1	nrcb2	f115w	clear
jw10002001001_01101_00006_nrcb4_uncal.fits	1	nrcb4	f115w	clear
jw10002001001_01101_00006_nrcb1_uncal.fits	1	nrcb1	f115w	clear
jw10002001001_01101_00006_nrcblong_uncal.fits	1	nrcblong	f444w	clear
jw10002001001_01101_00001_nrcb2_uncal.fits	1	nrcb2	f115w	clear
jw10002001001_01101_00005_nrcb4_uncal.fits	1	nrcb4	f115w	clear
jw10002001001_01101_00001_nrcblong_uncal.fits	1	nrcblong	f444w	clear
jw10002001001_01101_00001_nrcb4_uncal.fits	1	nrcb4	f115w	clear
jw10002001001_01101_00006_nrcb3_uncal.fits	1	nrcb3	f115w	clear