**DataPY Planner Module Rules**

*Rules to be implemented in planner modules during Phase 1* *Build-time validation and DAG construction rules*

**Section 1: Fundamental Principles (Planner Aspects)**

**1.1 DAG Structure Requirements**

* ✅ **Job must be acyclic directed graph** with components as vertices
* ✅ **Two edge types**: data edges (carry PipelineData) and control edges (triggers)
* ✅ **Deterministic structure**: Given identical YAML, produce identical DAG

**Section 2: Control-Edge Validation**

**2.1 Edge Syntax Validation**

* ✅ **ok edge**: component\_a (ok) component\_b - validate component names exist
* ✅ **error edge**: component\_a (error) component\_b - validate component names exist
* ✅ **if edge**: component\_a (if1): "condition" component\_b - validate syntax and order
* ✅ **parallelise edge**: Must list ≥2 targets, validate all targets exist
* ✅ **synchronise edge**: Validate upstream parallel branches exist
* ✅ **subjob\_ok/subjob\_error**: Validate these reference actual subjobs

**2.2 Control Edge Connectivity Rules**

* ✅ **Mutual exclusivity validation**: Components can have both ok AND error edges (not mutually exclusive at planning time)
* ✅ **if condition syntax**: Validate expression syntax using expression evaluator
* ✅ **parallelise fan-out**: Must have ≥2 targets, validate all exist
* ✅ **Control edge never triggers data flow**: Only validates structure, not execution

**Section 3: Subjob Boundary Detection**

**3.1 Boundary Detection Algorithm**

* ✅ **Main subjob**: Default subjob for components not in parallel branches
* ✅ **parallelise creates subjobs**: Each target of parallelise edge starts new subjob
* ✅ **synchronise joins subjobs**: Validate synchronise targets match parallelise sources
* ✅ **Nested subjob detection**: Handle chained parallelise edges
* ✅ **Subjob ID assignment**: Assign unique subjob\_id to each component

**3.2 Orphan Component Handling**

* ✅ **Inheritance rules**: Components with no control edges inherit parent subjob
* ✅ **Reachability validation**: All components must be reachable from startable nodes
* ✅ **Error**: "UNCONNECTED\_NODE" for unreachable components

**Section 8: Build-Time Edge Case Detection**

**8.1 Structural Validation**

* ✅ **Cycle detection**: "CYCLE\_DETECTED" error for circular dependencies
* ✅ **Malformed YAML**: JSON-schema validation with path information
* ✅ **Component existence**: All referenced component types must exist in registry

**Section 10: Data-Flow Structure Validation**

**10.1 Port Connectivity Validation**

* ✅ **Valid port names**: All data connections use ports declared in registry
* ✅ **allow\_multi\_in validation**: Components with false must have exactly one inbound per port
* ✅ **Wildcard port resolution**: Expand input\_*/output\_* to concrete ports
* ✅ **Port sorting**: Sort ports alphanumerically for deterministic hashing

**10.2 Data Edge Structure**

* ✅ **Data edge syntax**: component\_a.port\_x -> component\_b.port\_y
* ✅ **Port existence**: Both source and target ports must exist in registry
* ✅ **Connection validation**: Ensure valid source→target port combinations

**Section 11: Joblet Integration (Build-Time)**

**11.1 Joblet Expansion Process**

* ✅ **Detection**: Identify components with type: "@joblet\_name"
* ✅ **Joblet loading**: Load joblet YAML from registry and filesystem
* ✅ **Hash validation**: Ensure file hash matches registry hash
* ✅ **Component renaming**: callerName.\_\_internal\_component
* ✅ **Global prefixing**: callerName.\_\_internal\_\_GLOBAL\_NAME
* ✅ **Connection rewriting**: Update internal connections with new names

**11.2 Joblet Validation Rules**

* ✅ **Recursion detection**: Prevent infinite expansion (>20 levels)
* ✅ **Name collision detection**: Ensure no duplicate component names after expansion
* ✅ **Port mapping validation**: Validate joblet input/output ports exist
* ✅ **Proxy node creation**: Create proxy nodes for port mapping

**11.3 Joblet Boundary Rules**

* ✅ **Subjob creation**: Each joblet invocation = new subjob root
* ✅ **Control edge inheritance**: Joblet inherits caller's incoming control edges
* ✅ **Port exposure validation**: Validate declared inputs/outputs in joblet YAML

**Section 12: Iterator Component Detection**

**12.1 Iterator Identification**

* ✅ **Type detection**: Identify type: "iterator" components
* ✅ **Port validation**: Validate iterator input/output ports
* ✅ **Body detection**: Identify downstream components that form iterator body
* ✅ **Special edge validation**: Validate iteration\_ok, iteration\_error, iterator\_complete syntax

**Section 13: Startup Component Detection**

**13.1 STARTABLE Identification**

* ✅ **Registry flag**: Components with startable = 1 in registry
* ✅ **No control edges**: Components with no inbound control edges after joblet expansion
* ✅ **Validation**: Ensure at least one STARTABLE component exists
* ✅ **Error**: "NO\_STARTABLE\_COMPONENTS" if none found

**Section 15: Planner Validation Matrix**

**15.1 Structural Validations (Unit Tests)**

| **Rule** | **Validation** | **Expected Error** |
| --- | --- | --- |
| Cycle detection | Circular control/data edges | CYCLE\_DETECTED |
| Port connectivity | Invalid port references | INVALID\_PORT\_CONNECTION |
| Component existence | Non-existent component types | COMPONENT\_NOT\_FOUND |
| Startable detection | No entry points | NO\_STARTABLE\_COMPONENTS |
| Wildcard expansion | input\_*/output\_* resolution | Concrete ports created |
| Joblet validation | Invalid joblet references | JOBLET\_NOT\_FOUND |
| Recursion detection | Nested joblets >20 levels | JOBLET\_RECURSION\_DEPTH |
| Global references | Invalid component\_\_GLOBAL | INVALID\_GLOBAL\_REFERENCE |

**15.2 Integration Validations**

| **Scenario** | **Test Type** | **Expected Outcome** |
| --- | --- | --- |
| Joblet expansion | Integration | Prefixed components in DAG |
| Subjob boundary detection | Integration | Correct subjob\_id assignment |
| Complex control flows | Integration | Valid DAG structure |
| Port wildcard resolution | Integration | Concrete ports materialized |

**Addendum: Planner-Specific Clarifications**

**A1. Joblet Error Structure Validation**

* ✅ **Internal error edges**: Validate joblet internal error handling structure
* ✅ **Error propagation paths**: Ensure error edges have valid targets within joblet

**A2. die\_on\_error Planning**

* ✅ **Component-level validation**: Validate die\_on\_error flag per component
* ✅ **No inheritance validation**: Ensure no invalid inheritance patterns in YAML

**A3. Control Edge Structure**

* ✅ **SKIPPED edge validation**: Validate that SKIPPED components don't define control edges (planning warning)

**A4. Iterator Structure Validation**

* ✅ **Global mode validation**: Validate accumulate/reset/override modes in iterator config
* ✅ **Body structure**: Validate iterator body forms valid sub-DAG

**A5. Global Variable Reference Validation**

* ✅ **Reference syntax**: Validate component\_\_GLOBAL\_VAR syntax
* ✅ **Component existence**: Ensure referenced components exist in DAG
* ✅ **Logical dependency**: Warn if global reference creates implicit dependencies not in control flow

**Implementation Priority for Planner**

**Phase 1A: Core Structure (Days 1-2)**

1. Basic DAG building (components + edges)
2. Cycle detection
3. Port connectivity validation
4. Component existence validation

**Phase 1B: Advanced Structure (Days 3-4)**

1. Subjob boundary detection
2. Wildcard port resolution
3. STARTABLE component detection
4. Control edge syntax validation

**Phase 1C: Joblet Integration (Days 5-6)**

1. Joblet detection and loading
2. Component/global prefixing
3. Recursion and collision detection
4. Proxy node creation

**Phase 1D: Comprehensive Validation (Days 7-8)**

1. Expression syntax validation
2. Global variable reference validation
3. Iterator component detection
4. Complete validation error reporting

**These rules focus on build-time validation and DAG construction. They ensure the DAG is structurally valid and ready for execution by the engine.**