Redfish for Factory Automation Work in Progress

DMTF Redfish Forum

Version WIP80 - June, 2024

Disclaimer

- The information in this presentation represents a snapshot of work in progress within the DMTF.
- This information is subject to change without notice. The standard specifications remain the normative reference for all information.
- For additional information, see the DMTF website: www.dmtf.org

Introduction

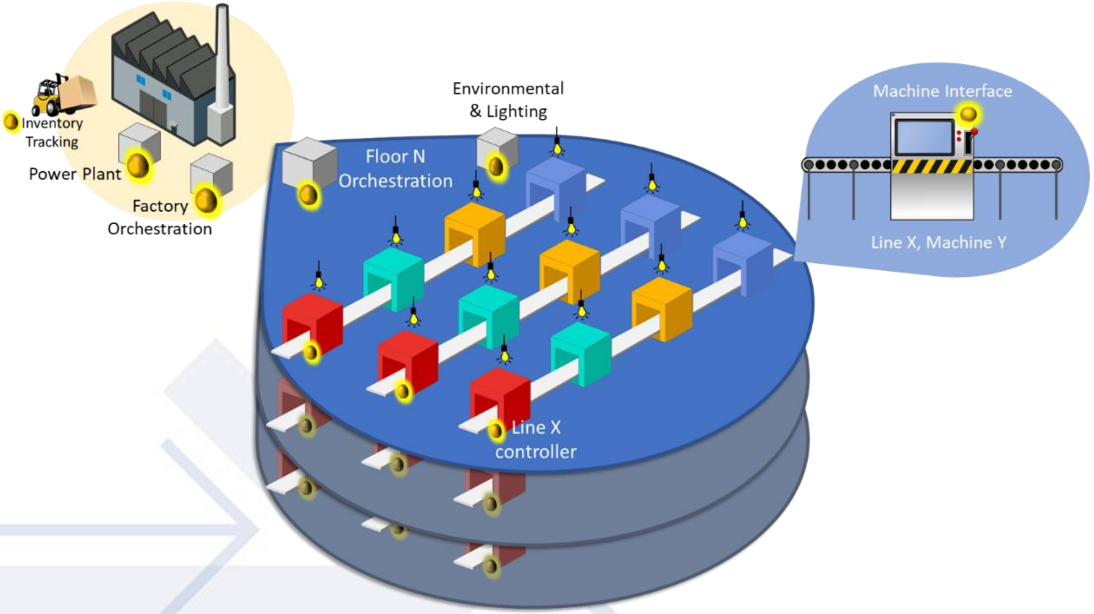
- DMTF Redfish Forum received a technology submission from PICMG proposing additions to the Redfish data model to support factory automation and Industrial “Internet of Things” (IIoT)
- The Redfish Forum has held discussions with members of PICMG to review their submission and provide feedback
- This work in progress bundle reflects the results of this collaboration with the intent to gather industry feedback and quickly bring this support into the Redfish standard

Context

- Factory 4.0 represents a significant shift in factory automation from the current state. Key attributes are:
  - Convergence of Information Technology and Operations Technology.
  - Internet of Things (smart sensors and effecters)
  - Big data and analytics
  - Custom and reconfigurable machinery
  - Cyber-physical such as digital twinning
- This proposal extends the Redfish data model to support Factory 4.0 automation tasks in two main areas:
  - Robust job management suitable for factory scheduling
  - Composable hardware configuration through automation building blocks

ENHANCEMENTS TO JOB MANAGEMENT

Factory Context for Redfish Job Service

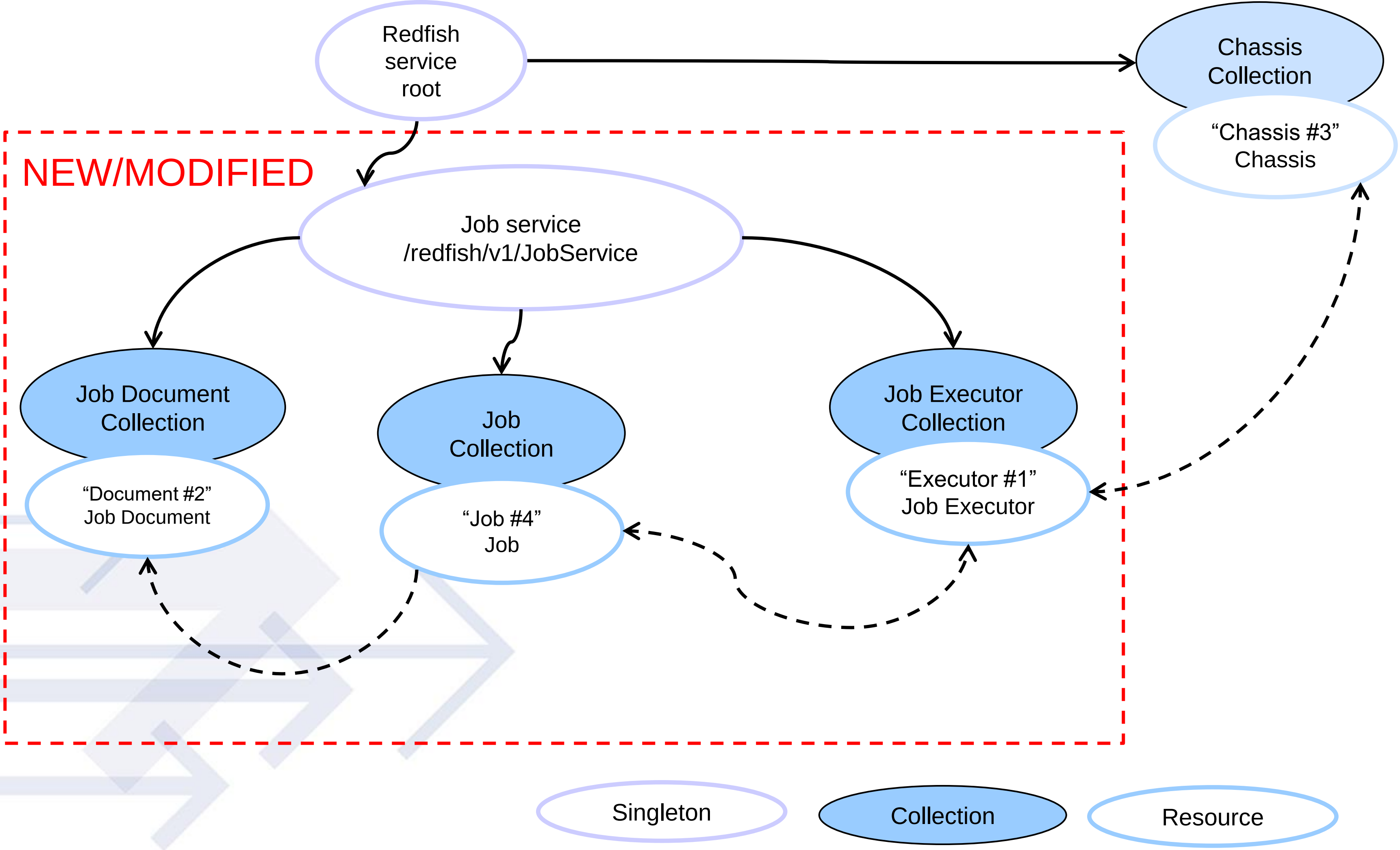


- Golden dots show potential locations of Redfish services
- Services operate in a hierarchy
  - Factory orchestration ->
    - Floor Orchestration ->
      - Line Control ->
        - Machine Control
- Each level in the hierarchy may have a 1 to many relationship
- Jobs started at higher levels of the hierarchy may spawn many jobs on lower levels of the hierarchy
- Debugging requires the ability to backtrack

Assumptions on usage

- New jobs consist of requests to execute factory “recipes” that enable configuration and operation of factory equipment
  - Factory equipment must be configured properly to run the jobs (e.g. correct tooling and ingredients)
  - Jobs are interpreted by the equipment’s executor(s) to coordinate the operation of **AutomationNodes** and other endpoints
- Jobs are created through the *SubmitJob* action on the related **JobDocument** resource (recipe)
  - Properties within the recipe describe optional and mandatory parameters for the *SubmitJob* action
- Job lifecycle is controlled by **JobService** policies and actions posted to the **JobService** or the specific **Job** instance

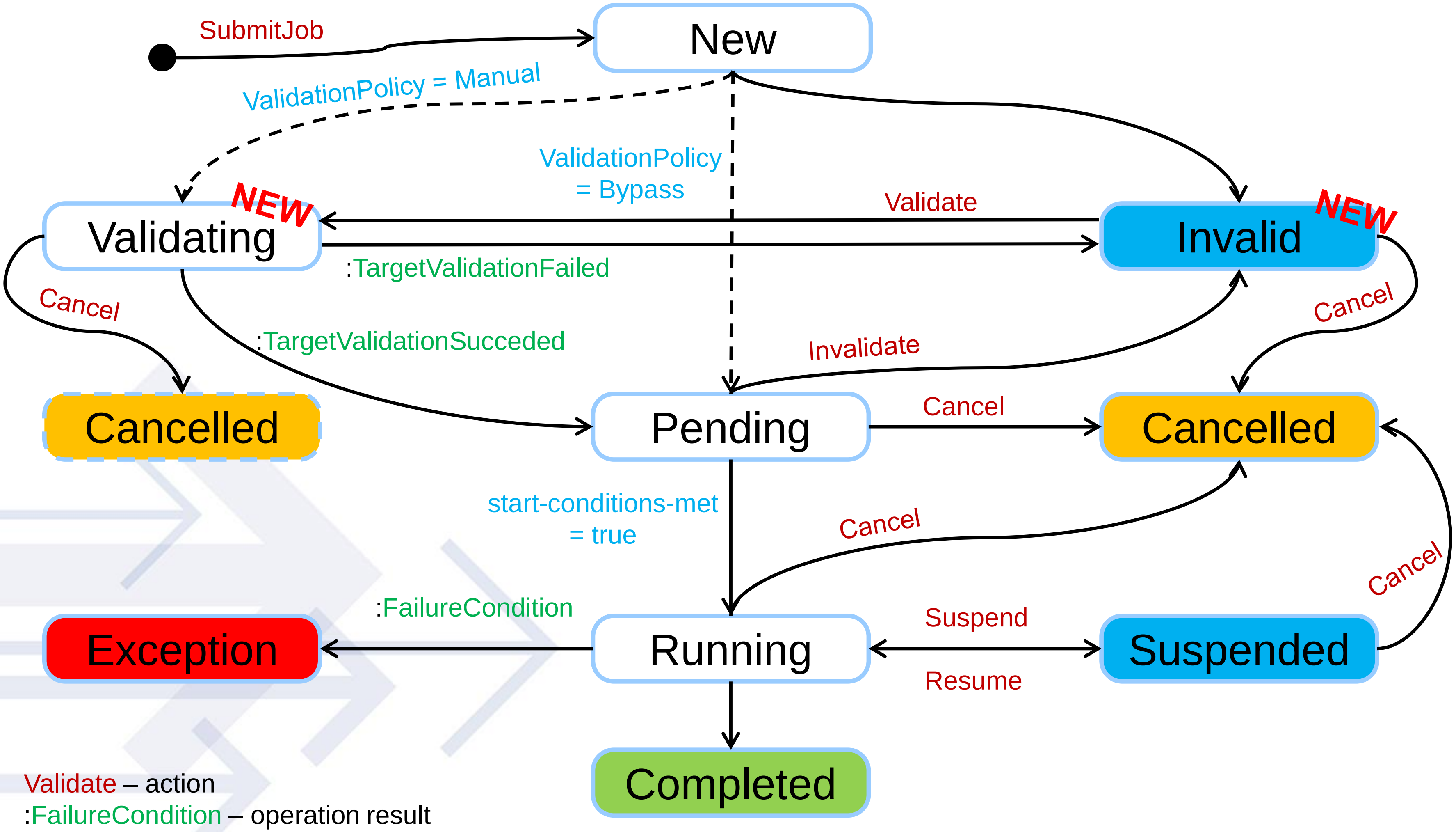
Updated Job Service Architecture



Additions to Redfish Job data model

- **NEW JobDocument**
  - New schema that defines the recipe to be executed
  - Describes required and optional parameters for job creation
- **NEW JobExecutor**
  - Describes a resource capable of running jobs of a specific type
- Additions to **Job** for document-based jobs
  - A **Job** resource represents an instance of a “recipe”
  - Lifecycle of the job follows a defined state machine
  - Lifecycle of the job may be controlled through actions

Document-Based JobState Lifecycle



SubmitJob

New

ValidationPolicy = Manual

ValidationPolicy = Bypass

NEW

Validating

Validate

Invalid

NEW

:TargetValidationFailed

Cancel

Cancel

:TargetValidationSucceded

Invalidate

Cancelled

Pending

Cancel

Cancelled

start-conditions-met = true

Cancel

Cancel

:FailureCondition

Exception

Running

Suspend

Resume

Suspended

Completed

Validate – action
:FailureCondition – operation result
ValidationPolicy=true - condition

Job Validation

- There are two validation steps: source validation, and target validation
- Source validation occurs during the **SubmitJob** action
    - Source validation checks for:
        - Malformed data or invalid parameters
        - No executors that can run the job
    - The action will return with an error indicating the issue
    - On error, no **Job** will be created
- Target validation may optionally be required for a given **JobService** based on the value of *ValidationPolicy*
    - Target validation checks the ability of the equipment to execute the job based on the machine's current state
    - Target validation checks for items like proper machine configuration and proper ingredients being present
    - If a job fails target validation, it will transition to the "Invalid" state
        - An operator may then reconfigure the equipment and attempt to revalidate the job so that it can be executed

JobDocument example

```
{
    "@odata.type": "#JobDocument.v1_0_0.JobDocument",
    "Id": "Recipe1",
    "Name": "Sweet candy",
    "Description" : "Produce specified amount of the sweet cand
    "Actions": {
        "#JobDocument.SubmitJob": {
            "target": "/redfish/v1/JobService/Documents/Recipe1/Actions/JobDocument.SubmitJob"
        }
    },
    "DocumentType" : "Script",
    "Version" : "1.0.0-20210215",
    "CreationTime" : "2022-01-16T04:14:33+06:00",
    "DocumentDataURI" : "https://recipestore.vendor.com/recipes?Action=Download&RecipeId=SweetCandy",
    "Status": {
        "State": "Enabled",
        "Health": "OK"
    },
```

The SubmitJob action – used to create Jobs related to this JobDocument

Where the document was obtained from

JobDocument example (continued)

```
     "ParameterMetadata": [
        {
            "Name" : "AmountPieces",
            "DataType" : "Number",
            "MinimumValue": 0,
            "Required": true,
            "Description": "something",
            "ValueHint": "10000"
        },
        {
            "Name": "Flavors",
            "AllowableValues": ["Orange","Strawberry","Lemon","Lime","Raspberry","Vanilla"],
            "DataType": "String",
            "Required": true,
            "Description": "The flavors of the candy.  One or two choices may be given",
            "ValueHint": "Orange"
        },
        {
            "Name":"Shape",
            "AllowableValues": ["Cylinder","Sphere","Dome","Cone"],
            "DataType" : "String",
            "Required": true,
            "Description": "The shape of the candy.",
            "ValueHint": "Dome"
        }
    ],
    "Links" : {"SupportedExecutors": [{"@odata.id":"/redfish/v1/JobService/Executors/1"}]    },
    "@odata.id": "/redfish/v1/JobService/Documents/Recipe1"
}
```

Metadata that describes parameters that are expected/accepted during the SubmitJob action.
Every JobDocument may have different required and optional parameters expected at when submitting a job.

A list of executors that will work with this JobDocument. It is expected that the service will complete this collection.

Job (updated) example

```
{
    "@odata.id": "/redfish/v1/JobService/Jobs/1",
    "@odata.type": "#Job.v1_3_0.Job",
    "Id": "1",
    "Name": "Document-based Job",
    "Description" : "This job was created using a the SubmitJob action on a JobDocument resource",
    "Actions": {
        "#Job.Cancel": {"target":"/redfish/v1/JobService/Jobs/1/Actions/Job.Cancel”},
        "#Job.Invalidate": {"target": "/redfish/v1/JobService/Jobs/1/Actions/Job.Invalidate"},
        "#Job.Resubmit": {"target": "/redfish/v1/JobService/Jobs/1/Actions/Job.Resubmit"},
        "#Job.Resume": {"target": "/redfish/v1/JobService/Jobs/1/A
        "#Job.ForceStart": {"target": "/redfish/v1/JobService/Jobs
        "#Job.Suspend": {"target": "/redfish/v1/JobService/Jobs/1/
        "#Job.Validate": {"target": "/redfish/v1/JobService/Jobs/1/Actions/Job.Validate"},
    },
    "JobType": "DocumentBased",
    "CreationTime" : "2022-01-16T04:14:33+06:00",
    "JobPriority": 0,
    "PercentComplete" : 25,
    "LastUpdateTime" : "2022-01-16T04:15:31+06:00",
    "EstimatedCompletionTime"  : "2022-01-16T04:18:31+06:00",
    "JobState": "Pending",
```

Actions to control the lifecycle of the job

Job type. DocumentBased jobs use the new behavior described in this proposal. Posted jobs behave as before.

The priority assigned for this job.

The state of the job. This field is updated as the job moves through its lifecycle.

Job (updated) example - continued

```
    "Parameters" : {
        "AmountPieces":10000,
        "Flavor":"Orange",
        "Shape":"Cylinder"
    },
    "StartTime" : "2022-01-16T04:14:33+06:00",
    "JobStatus": "OK",

    "Links" : {
        "ParentJob": { "@odata.id": "https://192.168.0.1/redfish/v1/JobS
        "SubsidiaryJobs": [
            { "@odata.id": https://192.168.0.3/redfish/v1/JobService/Jobs/1 }
        ],
        "JobDocument" : { "@odata.id" : "/redfish/v1/JobService/Document
        "Executor" : { "@odata.id" : "/redfish/v1/JobService/Executors/1
        "PreferredExecutors" : [
            { "@odata.id" : "/redfish/v1/JobService/Executors/1" }
        ],
        "ValidatedExecutors" : [{
            "@odata.id" : "/redfish/v1/JobService/Executors/1" }
        ]
    }
}
```

The parameters specified when the job was submitted. These may be different for jobs based on different job documents. The job document ParameterMetadata field specifies which parameters should be present for the job.

If the job was created by another job, contains a link to the job. Can be a remote or local URI

A collection of URIs to the jobs that were created by this job. Can be remote or local URIs

A link to the Executor that is executing this job

Links to preferred executors that were specified at job creation

Links to executors that this job has been validated to work on.

Job excerpt

- The job resource now supports an excerpt that can be used to gather status information without getting the entire Job resource
- Properties included in the excerpt:
    - *EstimatedCompletionTime*
    - *JobState*
    - *PercentComplete*

JobExecutor example

```
{
    "@odata.type": "#JobExecutor.v1_0_0.JobExecutor",
    "Id": "1",
    "Name": "Main Job Executor",
    "Description" : "Executor for Document-based Jobs",
    "ExecutorName" : "Script",
    "Status": {
        "State": "Enabled",
        "Health": "OK"
    },
    "MaximumConcurrentJobs":1,
    "Links" : {
        "ExecutingJobs" : [
            { "@odata.id" : "/redfish/v1/JobService/Jobs/1" }
        ],
        "Chassis" : { "@odata.id" : "/redfish/v1/Chassis/Aggregator" },
        "ComputerSystem" : { "@odata.id" : "/redfish/v1/Systems/1" }
    },
    "@odata.id": "/redfish/v1/JobService/Executors/1"
}
```

Specifies type of jobs executed.

Collection of jobs that are currently executing

JobService (updated) example

```
{
    "@odata.id": "/redfish/v1/JobService",
    "@odata.type": "#JobService.v1_2_0.JobService",
    "Actions": {
        "@odata.type": "#JobService.v1_2_0.Actions",
        "#JobService.CancelAllJobs": {"target": "/redfish/v1/JobService/Actions/JobService.CancelAllJobs"},
        "#JobService.InvalidateAllJobs": {"target": "/redfish/v1/JobService/Actions/JobService.InvalidateAllJobs"},
        "#JobService.ResumeAllJobs": {"target": "/redfish/v1/JobService/Actions/JobService.ResumeAllJobs"},
        "#JobService.SuspendAllJobs": {"target": "/redfish/v1/JobService/Actions/JobService.SuspendAllJobs"}
    },
    "DateTime": "2022-01-1T02:00:00+06:00",
    "Id": "JobService",
    "JobDocuments": {
        "@odata.id": "/redfish/v1/JobService/Documents"
    },
    "JobExecutors": {
        "@odata.id": "/redfish/v1/JobService/Executors"
    },
    "ValidationPolicy" : "Bypass",
    "Jobs": {
        "@odata.id": "/redfish/v1/JobService/Jobs"
    },
```

Actions to control Jobs

Job documents serve as “templates” for creating new jobs

Executors associated with the service

Policy related to validation of Document-based jobs

Job resources tracked by the service

JobService (updated) example (continued)

```
    "Log": {
        "@odata.id": "/redfish/v1/JobService/LogService"
    },
    "Name": "Job Service",
    "ServiceCapabilities": {
        "MaxJobs": 60,
        "MaxSteps": 5,
        "Scheduling": true,
        "UserSpecifiedJobs": true,
        "DocumentBasedJobs": true
    },
    "ServiceEnabled": true,
    "Status": {
        "State": "Enabled",
        "Health": "OK"
    }
}
```

Capabilities of the **JobService**. New fields added to report whether handing of user-specified or document-based jobs is supported.

Proposed Additions to the JobEvent Message Registry

- Defines messages for Document-based Job related events
  - *JobValidationBypassed* - A job has been queued without validation
  - *JobValidationStarted*- A job validation has been started
  - *JobValidatedOK* - A job validation has completed
  - *JobValidateException* - A job validation has completed with warnings or errors

AUTOMATION NODES

Where does Redfish Industrial IoT fit in the Factory?

<u>Sensor Nodes</u>
- Smart Sensors report capabilities
- Exposed as Redfish **Sensor**

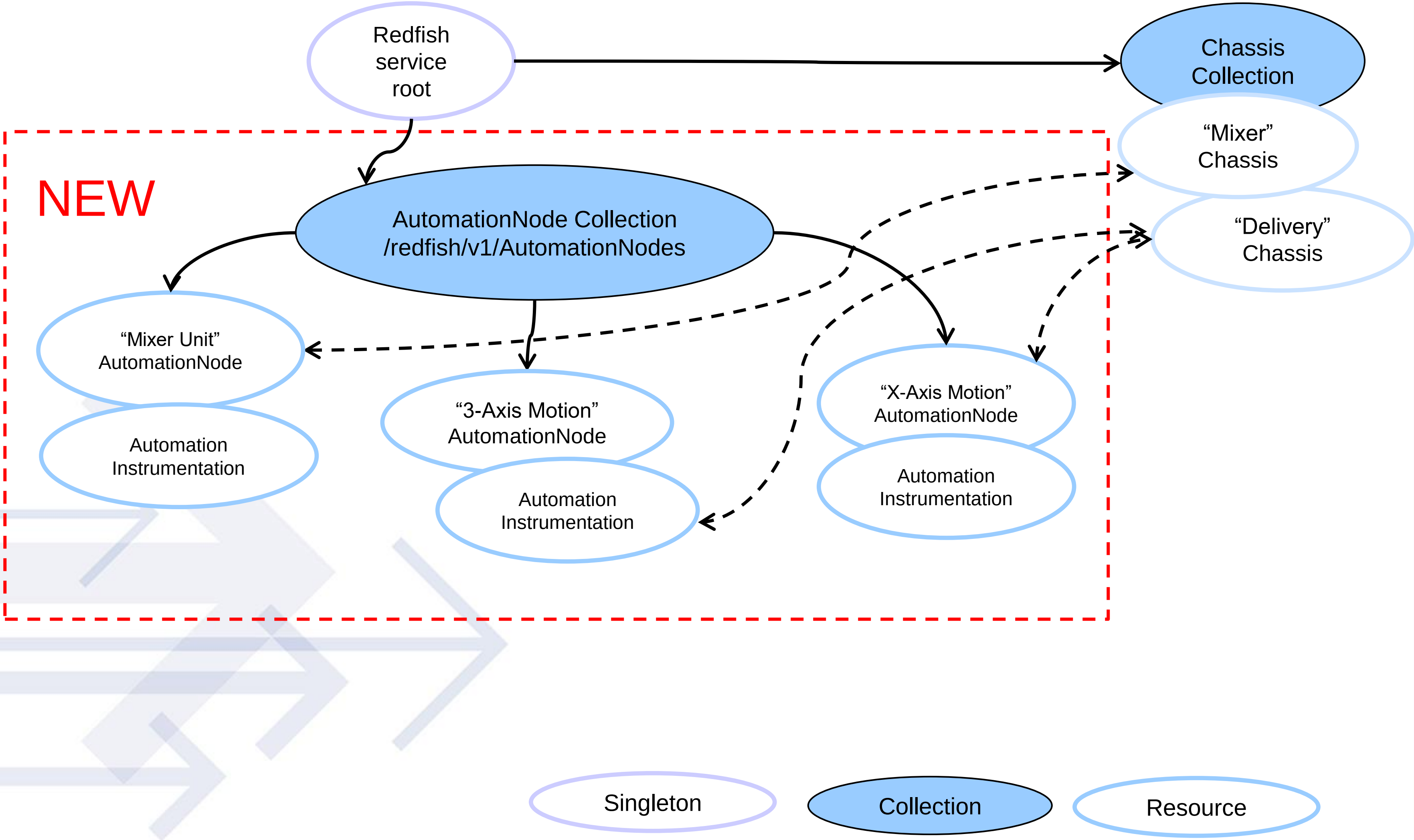
<u>Effector Nodes</u>
- Accept a *Setpoint*
- Controls Motion, Temperature, Pressure, etc.
- Exposed as Redfish **Control**

<u>Endpoint Controller</u>
- Includes multiple Sensor and Effector Nodes
- Synchronizes motion of multiple Effector Nodes
- Exposed as new Redfish **AutomationNode**

<u>Aggregator/Bridge</u>
- Can control multiple Endpoint Controllers
- Also exposed as an **AutomationNode**

Higher levels of integration supported to build a complete factory

AutomationNode Architecture


Redfish service root
Chassis Collection
"Mixer" Chassis
"Delivery" Chassis
NEW
AutomationNode Collection /redfish/v1/AutomationNodes
"Mixer Unit" AutomationNode
Automation Instrumentation
"3-Axis Motion" AutomationNode
Automation Instrumentation
"X-Axis Motion" AutomationNode
Automation Instrumentation
Singleton
Collection
Resource

NEW AutomationNode

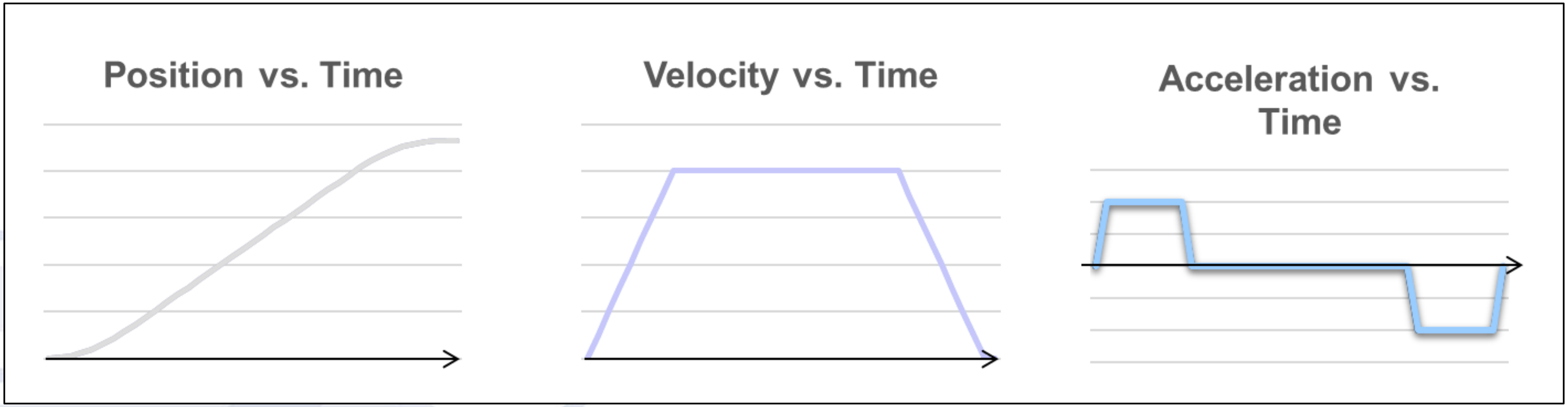
- Provides Smart Sensor or Smart Controller metrics and behaviors for industrial IoT and Factory Automation applications
- Compatible with PICMG IoT.1 device firmware specification
  - Other low-level solutions are possible
- *NodeType* explains purpose of the node and populates **Control** and **Sensor** data in the **AutomationInstrumentation** resource
  - Simple – simple set of control and associated sensor(s)
  - PID – PID-based control loop
  - MotionPosition, MotionVelocity – single axis synchronized robotic motion
  - MotionPositionGroup – multi-axis synchronized robotic motion
- *NodeState* shows the current state of the unit
  - Running, Idle, Waiting, Done, ConditionStop, ErrorStop
- Actions provide means to affect the state of the node
  - Start, Stop, Wait, Reset, SendTrigger

NEW AutomationNode, continued

- *MotionProfile* allows selection of profile motion types
- *AssertInterlock* provides means to manually trigger the device interlock
  - This could be modeled as an Action instead
  - Action becomes a POST to a specific URI with different permissions than a PATCH operation on the AutomationNode
- Complex factory equipment can be composed of multiple **AutomationNode** resource instances
  - Nodes can be “grouped” together to present unified data when applicable
  - For example, a multi-axis motion controller includes 3 single-axis nodes
  - Grouping allows for any operation to affect all axis at once (atomic) while providing individual configuration of the underlying single-axis nodes

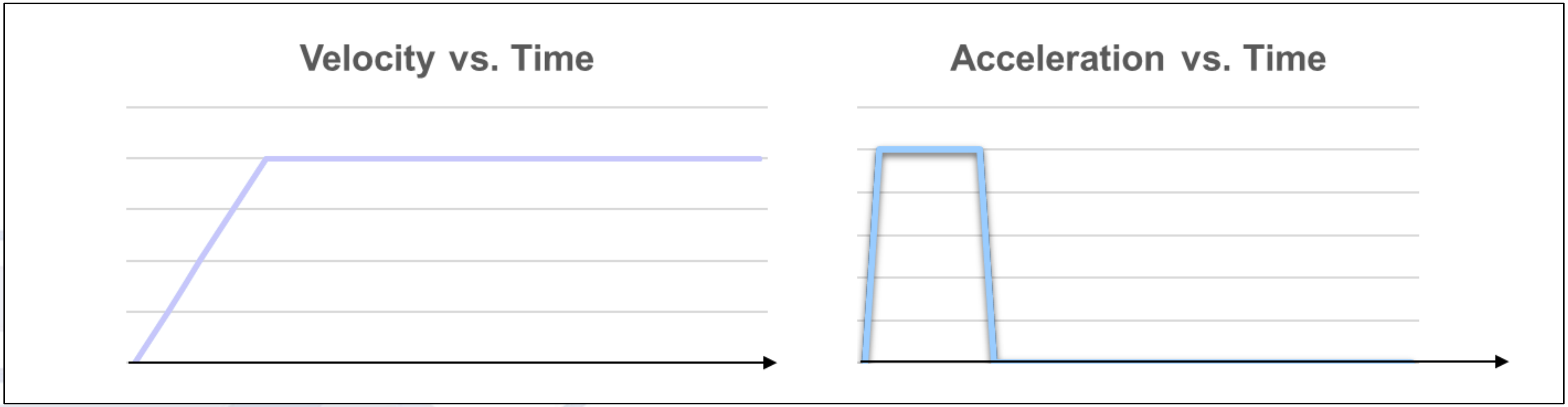
Motion Position NodeType

- Controls velocity and acceleration while moving to a specific position
- This mode is important for a variety of robotics operations
- Multiple axis can be synchronized together for complex motions

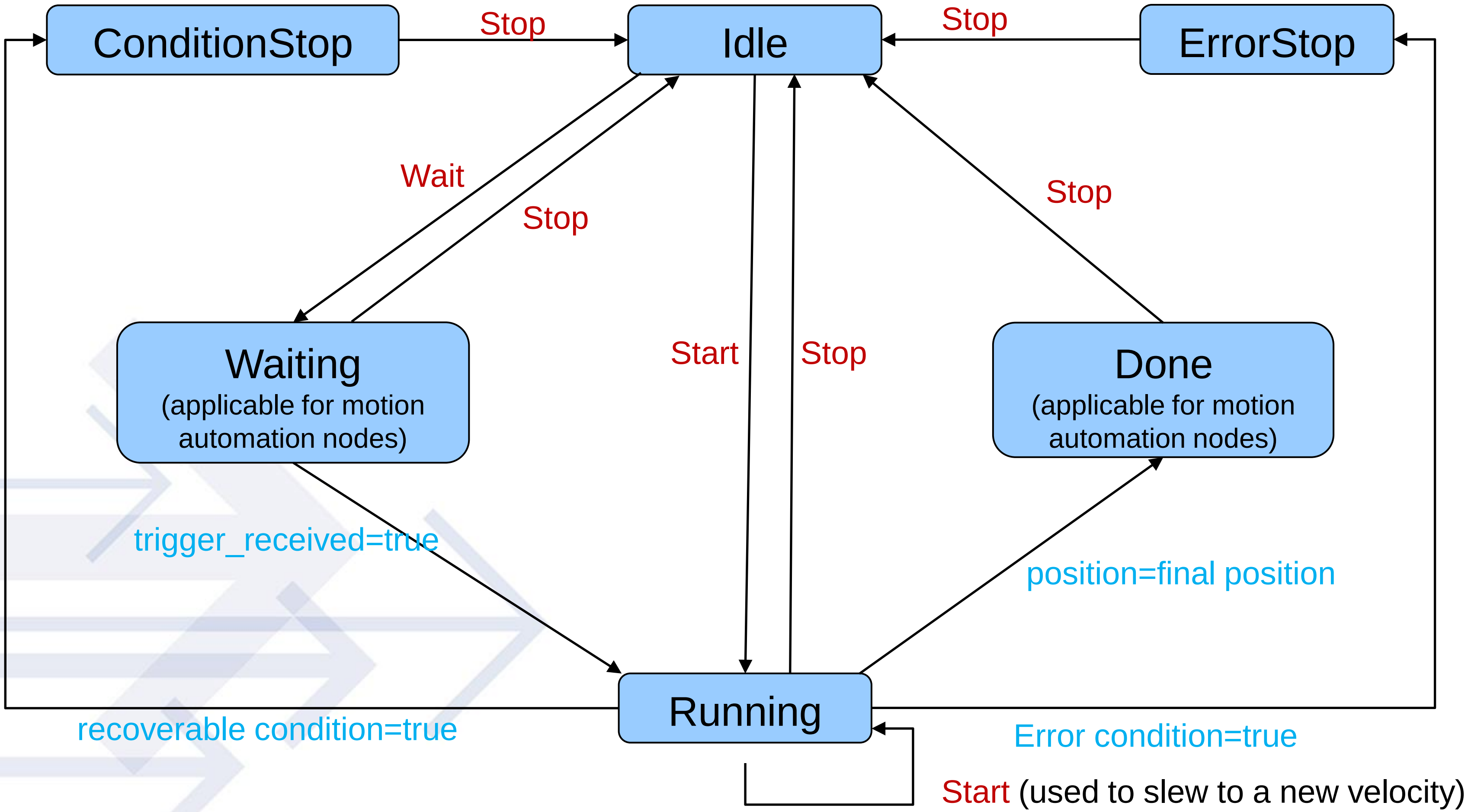


Motion Velocity NodeType

- Controls an output to a constant velocity (no final position)
- This mode is important for a variety of robotics control operations
- Multiple axis can be synchronized together for complex motions



NodeState state diagram (applies to any NodeType)

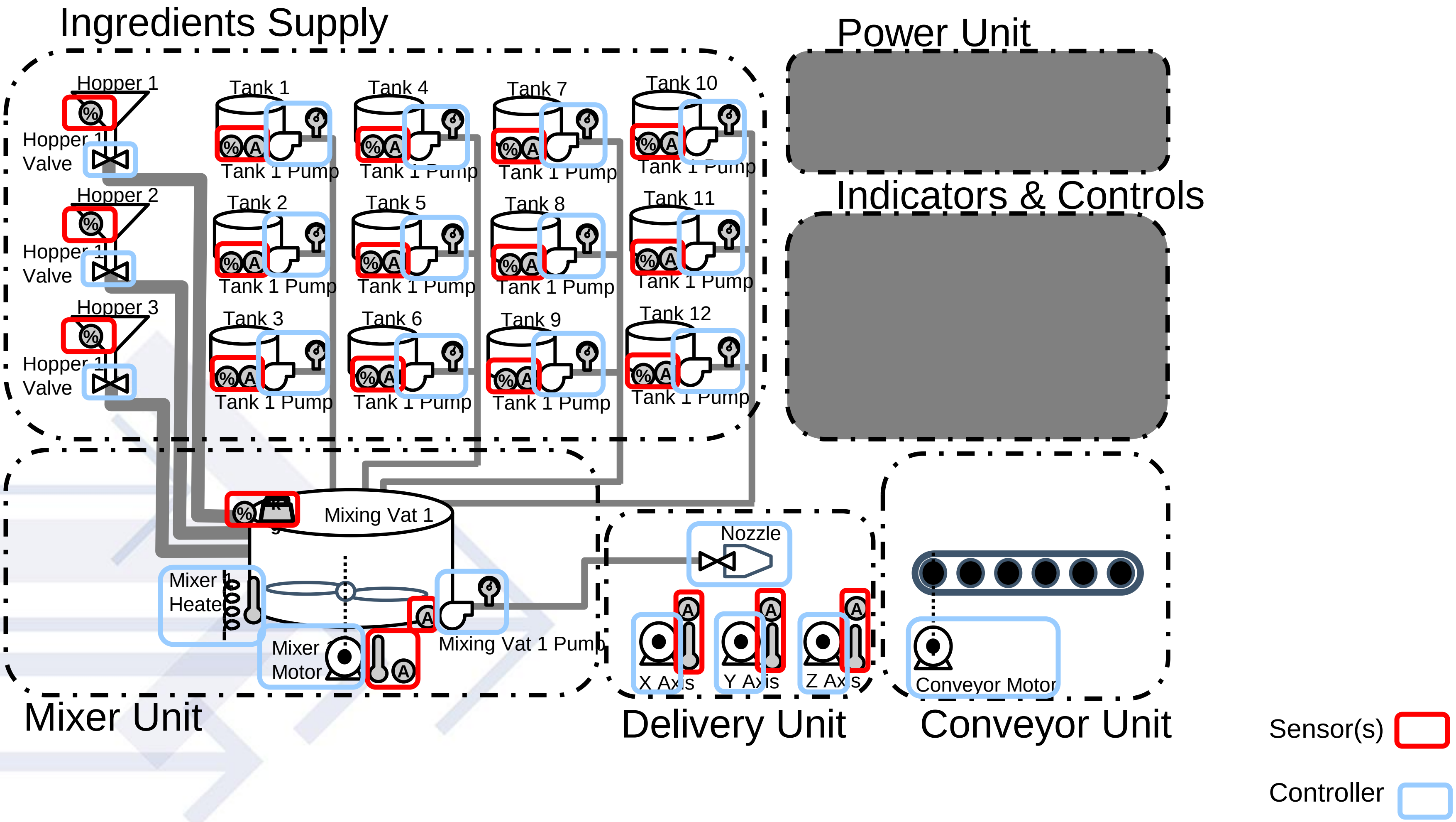

ConditionStop
Stop
Idle
Stop
ErrorStop
Wait
Stop
Stop
Start
Stop
Waiting
(applicable for motion automation nodes)
Done
(applicable for motion automation nodes)
trigger_received=true
position=final position
Running
recoverable condition=true
Error condition=true
Start (used to slew to a new velocity)

Red Text - actions
Blue Text - conditions

NEW AutomationInstrumentation

- Resource for monitoring the node and adjusting the controls
- Builds on excerpts of **Control** and **Sensor** instances
  - Details for each control or sensor, such as limits and alert thresholds, can be accessed by following *DataSourceUri* links
- Properties are populated depending on *NodeType*
  - Motion-related properties only appear for motion-based nodes
  - Additional multi-axis motion properties appear for motion “groups”
- Related sensor readings can also be populated for any *NodeType*
  - *TemperatureCelsius, Voltage, CurrentAmps*

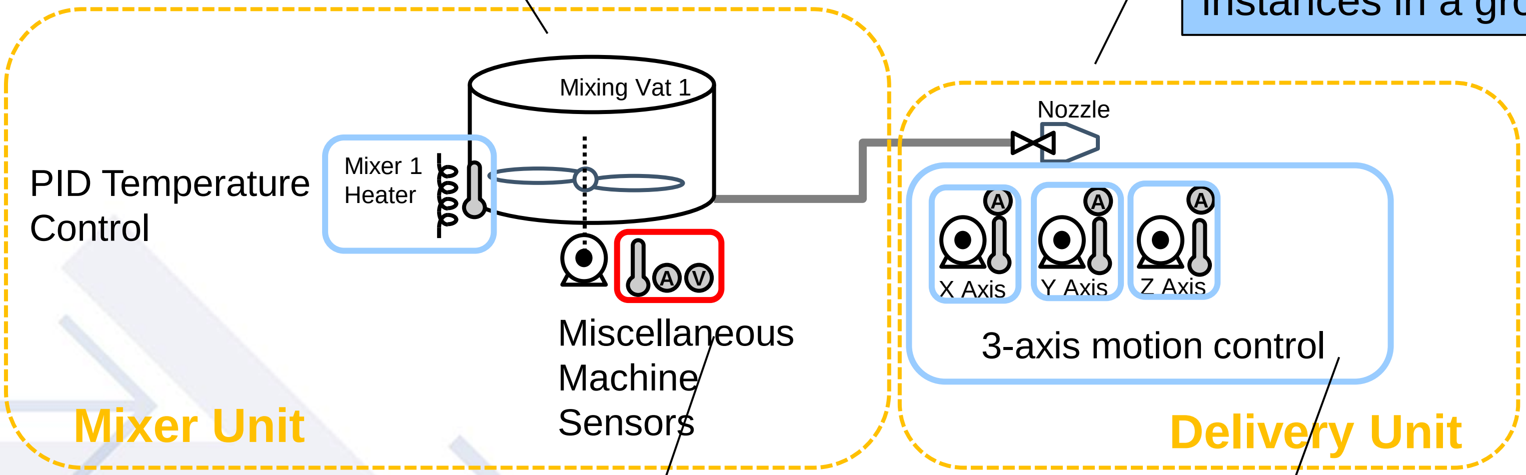
Representative Factory Machine (Candy Printer)



AutomationNode example: Mixer and Delivery Unit

Chassis with an **AutomationNode** and general sensors

Chassis with multiple **AutomationNode** instances in a group



General **Sensor** readings shown in **EnvironmentMetrics** for the **Chassis**

The 3-axis motion node **AutomationNode** builds upon a “group” of three additional nodes (X, Y, Z)

Sensor(s) 🟥

Automation Node ▭

Chassis ⬚

AutomationNode example

```
{
    "@odata.type": "#AutomationNode.v1_0_0.AutomationNode",
    "Id": "XAxisMover",
    "Name": "X-Axis positioner",
    "Description": "The X axis automation node for the machine.",
    "Status": {
        "State": "Enabled",
        "Health": "OK",
        "Conditions": []
    },
    "AssertInterlock": false,
    "NodeState": "Running",
    "NodeType": "MotionPosition",
    "MotionProfile": "Trapezoidal",
    "MotionAxis": "X",
    "Instrumentation": {
        "@odata.id": "/redfish/v1/AutomationNodes/XAxisMover/AutomationInstrumentation"
    },
    "Links": {
        "Chassis": [ {"@odata.id": "/redfish/v1/Chassis/3AxisMotion“ }
        ]
    },
    "@odata.id": "/redfish/v1/AutomationNodes/XAxisMover"
}
```

Motion-related properties allow selection of Profile types and axis alignment

AutomationInstrumentation example

```
{
    "@odata.type": "#AutomationInstrumentation.v1_0_0.AutomationInstrumentation",
    "Id": "Instrumentation",
    "Name": "Instrumentation for X-Axis Motion Controller",
    "Status": {
        "State": "Enabled",
        "Health": "OK",
        "Conditions": []
    },
    "NodeState": "Running",
    "PositionMeters": {
        "DataSourceUri": "/redfish/v1/Chassis/3AxisMotion/Controls/XOutput",
        "SetPoint": 0.75,
        "Reading": 0.51
    },
    "VelocityMpS": {
        "DataSourceUri": "/redfish/v1/Chassis/3AxisMotion/Controls/XVelocity",
        "Reading": 0.02
    },
    "AccelerationMpS2": {
        "Reading": 0.0
    },
    "@odata.id": "/redfish/v1/AutomationNodes/XAxisMover/AutomationInstrumentation"
}
```

Position-based motion allows a *SetPoint* for desired final position

Velocity and Acceleration **Control** excerpts provide pass-through **Sensor** readings but lack a *SetPoint*

Handling “pending” automation node operations

- Automation nodes may have their “next operation” queued up to begin immediately after the current operation is completed
  - While this is technically the next ‘step’ in a Job or similar operation, the difference is the timing of the sequence
  - A motion controller would immediately begin its next ‘move’, not wait in an idle state for the job service to send a command
- Proposal uses a new *PendingSetPoint* on any **Control**
  - The client performs a PATCH on *SetPoint*, the service responds by populating *PendingSetPoint* containing the new desired value
  - Once the current operation is complete, the value of *SetPoint* changes, and the *PendingSetPoint* is removed from the payload
- This concept may be better handled using Redfish “Settings” resource
  - A subordinate Settings resource is present when “queued operations” are required by the node, along with a *@Redfish.Settings* annotation

Additions to Sensor and Control

- Added **Sensor** *ReadingType* and **Control** *ControlType* values
  - LinearPosition (m) and RotationalPosition (rad)
  - LinearVelocity (m/s) and RotationalVelocity (rad/s)
  - LinearAcceleration (m/s^2) and RotationalAcceleration (rad/s^2)
- New **Control** *SetPointType* for "Monitor"
  - Allows a **Control** instance to "pass-through" a related **Sensor** *Reading* without having a *SetPoint*
  - Used to populate Control excerpts where the implementation may be either a Control or a Sensor, depending on configuration
  - Enables clients to monitor a single *Reading* property, regardless of the presence of a *SetPoint*
    - Example: A motion controller may report Velocity, but don't allow adjustments
- Added *SetPointError* to report *Reading* difference from *SetPoint*
  - Terminology common for Control Loops

AutomationNode Event Message Registry Proposal

- AutomationNodeEvent Registry
  - To be developed (not included in the WIP80 release bundle)
- Define messages for AutomationNode related state changes
  - *NodeIdle*
  - *NodeWaiting*
  - *NodeRunning*
  - *NodeDone*
  - *NodeStoppedCondition*
  - *NodeStoppedError*
- Will need additional messages for error conditions, limits exceeded, interlocks, for example:
  - *InterlockAsserted, InterlockCleared*
  - *PositionUpperLimitExceeded, PositionLowerLimitExceeded*

Getting involved in Redfish

Redfish Standards page

- Schemas, Specs, Mockups, White Papers & more
- http://www.dmtf.org/standards/redfish

Redfish Developer Portal

- Redfish Interactive Resource Explorer
- Educational material, documentation & other links
- http://redfish.dmtf.org

Redfish User Forum

- User forum for questions, suggestions and discussion
- http://www.redfishforum.com

DMTF Feedback Portal

- Provide feedback or submit proposals for Redfish standards
- https://www.dmtf.org/standards/feedback

DMTF Redfish Forum

- Join the DMTF to get involved in future
- http://www.dmtf.org/standards/spmf

Q & A and Discussion