# SDPF SPECIFICATION — Industrial Controller PSR

**Prompt ID:** ic-psr-v1 **Version:** v1 **Status:** LOCKED **Style:** Dynamic Criticality Extension (Style 14) **Date:** 2026-04-19 **Problem Owner:** Hamza Abdullah, Architect

---

## PHASE 0 — VALIDATED PROBLEM STATEMENT

**Current State:** Industrial process controllers lose 100% of mission state on every restart or power failure with no automatic recovery, no component failover, and no criticality classification.

**Desired State:** Zero state loss across restarts with full mission persistence, automatic sensor and actuator failover, and continuous criticality-based response within pre-approved bounds.

**Gap:** 100% state loss on every restart — the capability is entirely absent.

**Impact:** Equipment damage and product loss from unplanned stops, safety risk from undetected process anomalies, and regulatory exposure from unauditable control events.

**Problem Statement:** "Industrial process controllers currently lose 100% of mission state on every restart or power failure with no automatic recovery, no component failover, and no criticality classification, against a target of zero state loss across restarts with full mission persistence, automatic failover, and continuous criticality-based response, resulting in equipment damage and product loss from unplanned stops, safety risk from undetected anomalies, and regulatory exposure from unauditable control events."

**Validation:** T-1 Observable ✓ T-2 Bounded ✓ T-3 Cause-Free ✓ T-4 Solution-Free ✓ **Phase 0 Status:** COMPLETE

---

## S-01 — STYLE DECLARATION

**Style:** 14 — Dynamic Criticality Extension **Justification:** The Industrial Controller PSR's permitted response set changes at runtime based on measured process state. At NOMINAL criticality the controller runs PID control autonomously. At ADVISORY and ELEVATED levels it continues control while alerting operators. At CRITICAL level it holds last safe position and requires operator authorisation to resume. At EMERGENCY level it initiates full safe shutdown. All escalation and de-escalation rules are structural — bound to sensor readings and defined thresholds, not to operator discretion at runtime. No human escalation paths exist at execution time.

---

## S-02 — STYLE CONTEXT

**Domain:** Mission-critical industrial process control **System Boundary:** The Industrial Controller PSR reads from primary and backup sensors, drives primary and backup actuators within defined output ranges, persists full mission state to disk on every cycle and on every criticality transition, restores state on restart, reports status to SCADA over TLS, and runs continuously until manually terminated or an unrecoverable lethal condition is reached.

**Supported Process Types:**

- Chemical processing temperature control
- Manufacturing assembly line automation
- Power plant turbine monitoring
- Water treatment chemical dosing
- Oil refinery valve control

**Problem Statement Reference:** The current state component maps to Processing Rules (S-05). The desired state maps to Output Guarantees (S-06). The gap drives all CRITICAL requirements. The impact defines Exception Handling scope (S-07).

**Out of Scope:** Sensor hardware, actuator hardware, PLC firmware, SCADA server implementation, network infrastructure below the application layer, AI or machine learning components.

---

## S-03 — EXTERNAL CONTRACT

**Interface Type:** Python application — CLI and REST API **Data Serialisation:** JSON for all state persistence, SCADA reporting, and API responses **Version:** v1 **Protocol:** HTTPS with TLS for all SCADA communication; HTTP for local mock SCADA **Authentication:** HMAC-SHA256 signed state records using ICPSR_SECRET environment variable; operator commands authenticated by operator_id **Authorisation Token:** Required for resume from CRITICAL and EMERGENCY states

**Integration Points:**

| Integration | Direction | Protocol | Notes |
|---|---|---|---|
| Sensor network | Read | TCP/IP polling | Primary and backup per variable |
| Actuator network | Write | TCP/IP command | Within configured output range only |
| SCADA server | Write | HTTPS/TLS | Mock: http://localhost:8443 |
| State persistence | Read/Write | Local filesystem | Windows path: configured via CONFIG_PATH |
| Operator interface | Read/Write | REST API + CLI | localhost only |

**Problem Boundary Reference:** System boundary is scoped to the controller process running on the target machine. Hardware interfaces are out of scope.

---

## S-04 — INPUT CONTRACT

### Sensor Inputs (polled every cycle)

| Field | Type | Constraints | Required |
|---|---|---|---|
| `sensor_id` | string | Non-empty, matches config registry | Yes |
| `value` | float | Within physically valid range for sensor type | Yes |
| `unit` | string | One of: °C, PSI, L/min, m, pH, mS/cm, bool | Yes |
| `timestamp` | ISO 8601 datetime | Not older than stale_threshold_seconds | Yes |

| Field | Type | Constraints | Required |
|---|---|---|---|
| `status` | string | One of: OK, FAIL, STALE | Yes |
| `source` | string | One of: PRIMARY, BACKUP | Yes |

### Actuator Feedback Inputs (polled every cycle)

| Field | Type | Constraints | Required |
|---|---|---|---|
| `actuator_id` | string | Non-empty, matches config registry | Yes |
| `position` | float | 0.0–100.0 | Yes |
| `status` | string | One of: OK, FAIL, OFFLINE | Yes |
| `mode` | string | One of: AUTO, MANUAL | Yes |
| `timestamp` | ISO 8601 datetime | Not older than stale_threshold_seconds | Yes |

### Operator Command Inputs (REST API / CLI)

| Field | Type | Constraints | Required |
|---|---|---|---|
| `command` | string | One of: start, stop, safe, resume, acknowledge | Yes |
| `operator_id` | string | Non-empty | Yes |
| `target` | string | Process or actuator ID | Conditional |

| Field | Type | Constraints | Required |
|---|---|---|---|
| `authorisation_token` | string | HMAC-SHA256 verified token | Required for resume |

### Configuration File Inputs

| Field | Type | Constraints | Required |
|---|---|---|---|
| `process.target` | float | Within valid physical range for process type | Yes |
| `process.tolerance` | float | > 0 and < safety_limit - target | Yes |
| `process.safety_limit` | float | > target + tolerance | Yes |
| `process.type` | string | One of: TEMPERATURE, PRESSURE, FLOW, LEVEL, PH, CONDUCTIVITY | Yes |
| `process.pid.kp` | float | > 0 | Yes |
| `process.pid.ki` | float | >= 0 | Yes |
| `process.pid.kd` | float | >= 0 | Yes |
| `actuator.output_min` | float | >= hardware minimum | Yes |
| `actuator.output_max` | float | <= hardware maximum, > output_min | Yes |

| Field | Type | Constraints | Required |
|---|---|---|---|
| `actuator.safe_position` | float | Within output_min–output_max | Yes |
| `persistence.interval_seconds` | integer | 1–60 | Yes |
| `resilience.watchdog_timeout_seconds` | integer | 10–300 | Yes |
| `resilience.stability_window_minutes` | integer | 1–60 | Yes |

## S-05 — PROCESSING RULES

### PR-1: PSR Mission Loop

The controller shall run a continuous non-terminating mission loop. Each cycle shall execute the following steps in order:

```
1. Emit watchdog heartbeat
2. Read sensors (primary, fall back to backup on failure)
3. Read actuator feedback
4. Classify criticality level from current readings
5. Execute permitted response for current criticality level
6. Persist state to disk
7. Report status to SCADA
8. Sleep for configured cycle_interval_ms
9. Return to step 1
```

The loop shall not exit except on: operator stop command, EMERGENCY safe shutdown completion awaiting manual reset, or unrecoverable failure with all components failed and

no backup available.

### PR-2: PSR Formal Definition

```
PSR = (S, M, P, T, A, R)
```

| Symbol | Component | Implementation |
|---|---|---|
| S | State Space | Sensor readings, actuator states, criticality level, PID state, cycle count |
| M | Mission Function | maintain_process(target, tolerance) |
| P | Persistence Function | save_state() / load_state() |
| T | Transition Rules | read → classify → respond → persist → report |
| A | Adaptation Engine | switch_to_backup_sensor() / switch_to_backup_actuator() |
| R | Resilience Function | recover_from_failure() / resume_mission() |

### PR-3: Criticality Classification

On every cycle the controller shall classify current process state into exactly one criticality level using the state-escalation map defined in S-11. Classification shall complete within 200ms of sensor data receipt. The previous criticality level shall be retained until classification produces a different result.

### PR-4: Criticality-Gated Response

The controller's permitted response set is determined entirely by the current criticality

level as defined in S-11. No response outside the permitted set for the current criticality level shall be executed under any circumstances.

**PR-5: PID Control**

At NOMINAL and ADVISORY criticality the controller shall run a PID control loop:

```
error = target - current_reading
integral += error * dt
derivative = (error - last_error) / dt
output = (Kp * error) + (Ki * integral) + (Kd * derivative)
output = clamp(output, output_min, output_max)
```

PID output shall never exceed configured output_min or output_max regardless of computation result. At ELEVATED criticality PID output range shall be reduced to output_min–(output_max * 0.75). At CRITICAL and EMERGENCY PID output is suspended.

**PR-6: Sensor Failover**

[CRITICAL] If primary sensor read fails or returns STALE:

1. Switch to backup sensor within the same cycle
2. Log failure event with timestamp and sensor_id
3. Continue mission using backup sensor data
4. Include sensor_source: BACKUP in SCADA report

[CRITICAL] If both primary and backup sensors fail for any controlled variable:

1. Escalate criticality to ELEVATED minimum
2. Suspend PID output for that variable
3. Hold last known good actuator position

4. Alert operator immediately
5. Do not resume control until a functioning sensor is confirmed

**PR-7: Actuator Failover**

[CRITICAL] If primary actuator fails or reports FAIL status:

1. Switch to backup actuator within the same cycle
2. Transfer last output value to backup actuator
3. Log failure event with timestamp and actuator_id
4. Include actuator_source: BACKUP in SCADA report

[CRITICAL] If both primary and backup actuators fail:

1. Escalate criticality to CRITICAL minimum
2. Suspend all output
3. Alert operator immediately
4. Await operator authorisation for manual intervention

**PR-8: State Persistence**

[CRITICAL] System state shall be saved to disk:

- Every configured persistence interval (default: 10 seconds)
- On every criticality level transition
- On receipt of stop or safe command before shutdown

All state saves shall be atomic — written to a temporary file then renamed:

```
path = C:\Users\teach\Desktop\PythonCode\data\state.json
tmp  = C:\Users\teach\Desktop\PythonCode\data\state.json.tmp
1. Write full state to tmp
2. Rename tmp to path (atomic on Windows NTFS)
3. Log save event with timestamp and HMAC signature
```

State records shall be HMAC-SHA256 signed using ICPSR_SECRET.

**PR-9: State Restoration on Restart**

[CRITICAL] On startup the controller shall:

1. Attempt to load state from configured state file path
2. Verify the HMAC-SHA256 signature of the loaded state
3. If valid: restore all state variables and resume mission from saved criticality level
4. If invalid or absent: start clean session, log event, alert operator via SCADA
5. Verify sensor and actuator availability before resuming control output

**PR-10: Watchdog**

A watchdog thread shall run independently of the mission loop. If no heartbeat is received within watchdog_timeout_seconds:

1. Log watchdog timeout event
2. Attempt to load last saved state
3. Restart the mission loop thread
4. Resume mission from restored state
5. Alert operator via SCADA

**PR-11: SCADA Reporting**

The controller shall report status to SCADA_URL every configured reporting interval over HTTPS. If SCADA connection fails:

1. Switch to local-only operation immediately
2. Queue status reports up to queue_max_size
3. Retry with exponential backoff (base: 2 seconds, max: 60 seconds)
4. Flush queued reports when connection restores
5. Continue mission — SCADA unavailability shall never halt or pause the control loop

**PR-12: Safe State Procedure**

[CRITICAL] When safe state is required (EMERGENCY criticality or operator safe command):

1. Command all actuators to their configured safe_position
2. Suspend PID output
3. Set system_state to SAFE
4. Log safe state entry with timestamp, reason, and operator_id if operator-initiated
5. Report SAFE state to SCADA
6. Continue monitoring sensors
7. Halt all control output
8. Wait for manual operator reset with valid authorisation_token
9. Do not resume control until reset is acknowledged and confirmed

**PR-13: Windows Path Handling**

All file paths shall use Windows path conventions. The pathlib.Path library shall be used for all path construction to ensure cross-platform correctness within the Windows

environment. Hardcoded Unix paths (e.g. /var/lib/) shall not appear anywhere in the implementation.

---

## S-06 — OUTPUT GUARANTEES

### SCADA Status Report

Sent every reporting interval. All fields shall be present on every report.

```json
{
  "timestamp": "ISO 8601 UTC",
  "controller_id": "string",
  "process_id": "string",
  "mission": "string",
  "target": "float",
  "tolerance": "float",
  "current": "float",
  "unit": "string",
  "criticality_level": "NOMINAL | ADVISORY | ELEVATED | CRITICAL | EMERGENCY",
  "system_state": "RUNNING | SAFE | STOPPED",
  "actuator_output": "float",
  "sensor_source": "PRIMARY | BACKUP",
  "actuator_source": "PRIMARY | BACKUP",
  "uptime_seconds": "integer",
  "cycle_count": "integer",
  "last_state_save": "ISO 8601 UTC"
}
```

### Operator Alert

Generated on every criticality transition above NOMINAL and on every component failure.

```json
{
  "alert_id": "string — UUID4",
  "timestamp": "ISO 8601 UTC",
  "criticality_level": "string",
  "process_id": "string",
  "condition": "string — plain-language condition description",
  "current_reading": "float",
  "target": "float",
  "safety_limit": "float",
  "sensor_status": "string",
  "actuator_status": "string",
  "action_taken": "string — what the controller did automatically",
  "operator_action_required": "boolean",
  "authorisation_required": "boolean"
}
```

### Persistence Record

Written on every save. Signature covers all fields except hmac_signature itself.

```json
```

```
{
  "version": "1.0",
  "timestamp": "ISO 8601 UTC",
  "controller_state": {
    "process_id": "string",
    "target": "float",
    "tolerance": "float",
    "safety_limit": "float",
    "current_reading": "float",
    "unit": "string",
    "criticality_level": "string",
    "system_state": "string",
    "actuator_output": "float",
    "pid_integral": "float",
    "pid_last_error": "float",
    "active_sensor": "PRIMARY | BACKUP",
    "active_actuator": "PRIMARY | BACKUP",
    "stability_window_start": "ISO 8601 UTC | null",
    "cycle_count": "integer",
    "start_time": "ISO 8601 UTC",
    "last_update": "ISO 8601 UTC"
  },
  "sensor_readings": [
    {
      "sensor_id": "string",
      "value": "float",
      "unit": "string",
      "status": "string",
      "source": "string",
      "timestamp": "ISO 8601 UTC"
    }
  ],
```

```
  "actuator_states": [
    {
      "actuator_id": "string",
      "output": "float",
      "position": "float",
      "status": "string",
      "mode": "string",
      "timestamp": "ISO 8601 UTC"
    }
  ],
  "hmac_signature": "string — HMAC-SHA256 hex digest"
}
```

### API Status Response (GET /status) — HTTP 200

json

```
{
  "timestamp": "ISO 8601 UTC",
  "system_state": "RUNNING | SAFE | STOPPED",
  "criticality_level": "string",
  "process_id": "string",
  "current_reading": "float",
  "target": "float",
  "unit": "string",
  "actuator_output": "float",
  "active_sensor": "PRIMARY | BACKUP",
  "active_actuator": "PRIMARY | BACKUP",
  "uptime_seconds": "integer",
  "cycle_count": "integer",
  "scada_connected": "boolean"
}
```

## S-07 — EXCEPTION HANDLING

| Error Code | Condition | Controller Response | API Status |
|---|---|---|---|
| `SENSOR_PRIMARY_FAIL` | Primary sensor read failure | Switch to backup, log, alert, continue | N/A internal |
| `SENSOR_ALL_FAIL` | All sensors for variable failed | Hold position, escalate ELEVATED min, alert | N/A internal |
| `ACTUATOR_PRIMARY_FAIL` | Primary actuator failure | Switch to backup, log, alert, continue | N/A internal |
| `ACTUATOR_ALL_FAIL` | All actuators failed | Suspend output, escalate CRITICAL min, alert | N/A internal |
| `OUTPUT_CLAMPED` | PID output exceeds configured range | Clamp to output_min/max, log | N/A internal |
| `STATE_RESTORE_FAIL` | Saved state corrupt or invalid HMAC | Start clean session, log, alert operator | N/A internal |
| `STATE_WRITE_FAIL` | State file write failed | Retry once, log error, continue mission | N/A internal |
| `SCADA_UNREACHABLE` | SCADA connection failed | Queue reports, retry backoff, continue | N/A internal |
| `WATCHDOG_TIMEOUT` | No heartbeat within timeout | Restart mission loop, restore state | N/A internal |

| Error Code | Condition | Controller Response | API Status |
|---|---|---|---|
| `LETHAL_CONDITION` | Reading exceeds safety_limit | Immediate safe state, alert all | N/A internal |
| `AUTH_TOKEN_INVALID` | Operator token fails HMAC verification | Reject command, log attempt | HTTP 401 |
| `COMMAND_NOT_PERMITTED` | Command invalid for current system_state | Reject, return current state | HTTP 409 |
| `PROCESS_NOT_FOUND` | Unknown process_id in request | Reject request | HTTP 404 |
| `INVALID_INPUT` | Missing or invalid field | Reject with field-level error detail | HTTP 422 |

All API error responses use the SDPF standard error schema:

```json
{"error": {"code": "string", "message": "string", "field": "string | null"}}
```

---

## S-08 — TECHNICAL VERIFICATION GATE

**Environment:** Conda environment `in_control` — must be active before any file is modified or any command is run.

**HALT RULE:** If any CRITICAL check fails, stop immediately. Do not proceed to implementation. Resolve the failure and re-verify before continuing.

| Asset | Asserted Value | Verification Command |
|---|---|---|
| Conda environment | `in_control` | `conda info --envs` |
| Python version | 3.13.1 | `python --version` |
| pip | 26.0.1 | `pip --version` |
| requests | 2.32.3 | `pip show requests` |
| cryptography | 44.0.2 | `pip show cryptography` |
| pyyaml | 6.0.3 | `pip show pyyaml` |
| psutil | 7.2.2 | `pip show psutil` |
| pytest | 9.0.2 | `pip show pytest` |
| pytest-cov | 7.0.0 | `pip show pytest-cov` |
| stdlib modules | available | `python -c "import hmac, json, pathlib, threading, print('ok')"` |

| Asset | Asserted Value | Verification Command |
|---|---|---|
| ICPSR_SECRET | SET | `python -c "import os; print('SET' if os.getenv('ICPSR_SECRET') else 'NOT SET')"` |
| SCADA_URL | SET | `python -c "import os; print('SET' if os.getenv('SCADA_URL') else 'NOT SET')"` |
| CONFIG_PATH | SET | `python -c "import os; print('SET' if os.getenv('CONFIG_PATH') else 'NOT SET')"` |
| LOG_PATH | SET | `python -c "import os; print('SET' if os.getenv('LO else 'NOT SET')"` |
| Data dir writable | writable | `python -c "import pathlib; p=pathlib.Path(r'C:\Users\teach\Desktop\PythonCode p.mkdir(parents=True, exist_ok=True); print('ok')"` |
| OS | Windows 11 | `python -c "import platform; print(platform.platfor` |

**TVG STATUS: ALL CHECKS VERIFIED — LOCKED 2026-04-19**

---

## S-09 — VERIFICATION REQUIREMENTS

| Test ID | REQ-ID | Description | Pass Condition |
|---|---|---|---|
| T- | REQ- | Mission loop starts and | Loop executes without error for 60 seconds |

| Test ID | REQ-ID | Description | Pass Condition |
|---|---|---|---|
| 001 | 001 | runs continuously | |
| T-002 | REQ-002 | PID control at NOMINAL maintains setpoint | Actuator output adjusts to maintain target ± tolerance |
| T-003 | REQ-003 | PID output clamped to configured range | Output never exceeds output_min or output_max |
| T-004 | REQ-004 | PID range reduced at ELEVATED criticality | Output does not exceed output_max * 0.75 at ELEVATED |
| T-005 | REQ-005 | Primary sensor failure triggers backup switch | Backup sensor used within one cycle of primary failure |
| T-006 | REQ-006 | Both sensors failed: hold position, escalate ELEVATED | Last position held; ELEVATED criticality set; operator alerted |
| T-007 | REQ-007 | Primary actuator failure triggers backup switch | Backup actuator used; last output value transferred |
| T-008 | REQ-008 | Both actuators failed: output suspended, CRITICAL set | No output issued; CRITICAL set; operator alerted |
| T-009 | REQ-009 | State saved every configured interval with valid HMAC | State file present; HMAC verifies after every save |

| Test ID | REQ-ID | Description | Pass Condition |
|---|---|---|---|
| T-010 | REQ-010 | State save is atomic — no partial writes | Temp file rename confirmed; no corrupt state on simulated crash |
| T-011 | REQ-011 | State saved on criticality transition | Save occurs within one cycle of every criticality change |
| T-012 | REQ-012 | Restart restores state from last valid signed record | Post-restart state matches pre-crash state exactly |
| T-013 | REQ-013 | Corrupt state starts clean session and alerts | Clean session started; operator alert generated |
| T-014 | REQ-014 | Watchdog detects missing heartbeat and restarts loop | Loop restarted within watchdog_timeout + 5 seconds |
| T-015 | REQ-015 | SCADA failure: mission continues, reports queued | Control loop uninterrupted; queue grows during outage |
| T-016 | REQ-016 | SCADA restored: queued reports flushed | All queued reports delivered after reconnection |
| T-017 | REQ-017 | ADVISORY: operator alert generated | Alert with all required fields delivered to SCADA |
| T-018 | REQ-018 | CRITICAL: PID suspended, position held, operator alerted | No PID output; last position maintained; urgent alert sent |
| T-019 | REQ-019 | EMERGENCY: safe state initiated | All actuators at safe_position; SAFE state set; output halted |

| Test ID | REQ-ID | Description | Pass Condition |
|---|---|---|---|
| T-020 | REQ-020 | Operator safe command initiates safe state | Safe state within 3 seconds of command receipt |
| T-021 | REQ-021 | Resume from safe requires valid authorisation token | Rejected without valid token; accepted with valid token |
| T-022 | REQ-022 | De-escalation requires full stability window | Criticality does not reduce before stability_window_minutes elapsed |
| T-023 | REQ-023 | EMERGENCY de-escalation requires manual reset only | No automatic de-escalation from EMERGENCY under any condition |
| T-024 | REQ-024 | Lethal condition triggers immediate safe state | Safe state within one cycle of safety_limit breach |
| T-025 | REQ-025 | Stop command ends mission cleanly | Actuators to safe_position; state saved; process exits with code 0 |
| T-026 | REQ-026 | All file paths use Windows conventions | No Unix-style paths in any generated file or log |
| T-027 | REQ-027 | Full pipeline end-to-end | Start → NOMINAL → simulate anomaly → ADVISORY → ELEVATED → CRITICAL → safe → resume → stop — no errors |

## S-10 — TRACEABILITY MATRIX

| REQ-ID | Requirement Summary | Test IDs | Implementation Module |
|---|---|---|---|
| REQ-001 | PSR mission loop | T-001 | `core/mission_loop.py` |
| REQ-002 | PID control at NOMINAL | T-002, T-003 | `control/pid_controller.py` |
| REQ-003 | PID output clamping | T-003 | `control/pid_controller.py` |
| REQ-004 | Reduced PID range at ELEVATED | T-004 | `control/pid_controller.py` |
| REQ-005 | Sensor primary failover | T-005, T-006 | `sensors/adapter.py` |
| REQ-006 | All-sensor-fail handling | T-006 | `sensors/adapter.py`, `criticality/classifier.py` |
| REQ-007 | Actuator primary failover | T-007, T-008 | `actuators/adapter.py` |
| REQ-008 | All-actuator-fail handling | T-008 | `actuators/adapter.py`, `criticality/classifier.py` |
| REQ-009 | Periodic signed persistence | T-009 | `persistence/manager.py` |

| REQ-ID | Requirement Summary | Test IDs | Implementation Module |
|---|---|---|---|
| REQ-010 | Atomic state write | T-010 | `persistence/manager.py` |
| REQ-011 | Persistence on criticality transition | T-011 | `persistence/manager.py`, `criticality/classifier.py` |
| REQ-012 | State restore on restart | T-012, T-013 | `persistence/manager.py` |
| REQ-013 | Corrupt state clean start | T-013 | `persistence/manager.py` |
| REQ-014 | Watchdog restart | T-014 | `core/watchdog.py` |
| REQ-015 | SCADA failure local operation | T-015, T-016 | `reporting/scada_client.py` |
| REQ-016 | SCADA queue flush on restore | T-016 | `reporting/scada_client.py` |
| REQ-017 | Operator alert generation | T-017, T-018 | `alerts/dispatcher.py` |
| REQ-018 | CRITICAL level handling | T-018 | `criticality/classifier.py`, `control/pid_controller.py` |
| REQ-019 | EMERGENCY safe state | T-019, T-024 | `safety/safe_state.py` |

| REQ-ID | Requirement Summary | Test IDs | Implementation Module |
|---|---|---|---|
| REQ-020 | Operator safe command | T-020 | `safety/safe_state.py` |
| REQ-021 | Authorised resume from safe | T-021 | `auth/gate.py` |
| REQ-022 | De-escalation stability window | T-022 | `criticality/classifier.py` |
| REQ-023 | No auto de-escalation from EMERGENCY | T-023 | `criticality/classifier.py` |
| REQ-024 | Lethal condition response | T-024 | `safety/safe_state.py` |
| REQ-025 | Clean stop | T-025 | `core/mission_loop.py` |
| REQ-026 | Windows path conventions | T-026 | All modules |
| REQ-027 | Full pipeline end-to-end | T-027 | All modules |

## S-11 — CRITICALITY LEVEL DEFINITIONS

### Criticality Levels

| Level | Name | Meaning |
|---|---|---|
| 0 | NOMINAL | All readings within target ± tolerance. PID control active at full range. |
| 1 | ADVISORY | Reading outside tolerance but within safe operating range. Trend requires attention. |
| 2 | ELEVATED | Reading approaching safety limit or component failure detected. Reduced control range. |
| 3 | CRITICAL | Reading at or beyond safety threshold or both actuators failed. PID suspended. |
| 4 | EMERGENCY | Reading exceeds safety_limit (lethal condition) or operator safe command. Full safe shutdown. |

### State-Escalation Map

| Trigger Condition | Resulting Level |
|---|---|
| All readings within target ± tolerance, all components OK | NOMINAL |
| Any reading > target + (2 × tolerance) | ADVISORY minimum |
| Any reading > target + (4 × tolerance) | ELEVATED minimum |
| Any reading > 90% of (safety_limit - target) + target | CRITICAL minimum |

| Trigger Condition | Resulting Level |
|---|---|
| Any reading >= safety_limit | EMERGENCY — immediate |
| Primary sensor FAIL | ADVISORY minimum |
| Both sensors FAIL for any variable | ELEVATED minimum |
| Primary actuator FAIL | ADVISORY minimum |
| Both actuators FAIL | CRITICAL minimum |
| Operator safe command received | EMERGENCY |

### De-escalation Procedures

De-escalation requires ALL listed conditions to be satisfied. Never automatic on a single reading. Requires sustained stability over stability_window_minutes (default: 5 minutes).

### Permitted Response Set by Criticality Level

| Criticality | PID Active | PID Range | Actuator Output | Operator Alert | Auth Required | Safe State |
|---|---|---|---|---|---|---|
| NOMINAL | YES | Full range | YES | No | No | No |
| ADVISORY | YES | Full range | YES | YES | No | No |
| ELEVATED | YES | 75% of max | YES — reduced | YES — escalated | No | No |

| Criticality | PID Active | PID Range | Actuator Output | Operator Alert | Auth Required | Safe State |
|---|---|---|---|---|---|---|
| CRITICAL | NO | Suspended | Hold last position | YES — urgent | YES for resume | No |
| EMERGENCY | NO | Suspended | safe_position | YES — emergency | YES for reset | YES — mandatory |

**DYNE Rules (Style 14 Normative)**

**DYNE-1:** Criticality level shall be applied automatically on every cycle according to the state-escalation map above. No human intervention required for escalation.

**DYNE-2:** Conflicts between requirements at different criticality levels are resolved automatically by strictness priority: CRITICAL > REQUIRED > OPTIONAL.

**DYNE-3:** A state transition that would create a CRITICAL-CRITICAL conflict shall be treated as EMERGENCY and shall trigger the safe state procedure immediately.

**DYNE-4:** No human escalation paths exist at runtime. All possible state transitions and their implications are fully defined in this specification. Human intervention occurs only at specification time and for authorised resume/reset commands.

---

## S-12 — STATE-ESCALATION MAP

Defined in full in S-11. The escalation map above is the sole normative source for all criticality transitions. No transition occurs outside this map. The map is complete — every reachable state is covered.

---

### S-13 — DE-ESCALATION PROCEDURES

Defined in full in S-11. Key constraints:

- De-escalation from EMERGENCY requires manual operator reset with valid authorisation_token only
- No automatic de-escalation from EMERGENCY is permitted under any circumstances
- De-escalation from CRITICAL requires authorised operator confirmation
- De-escalation from ADVISORY and ELEVATED requires operator acknowledgement
- All de-escalation requires the full stability_window_minutes of sustained readings

---

## DATA STRUCTURES

```python
```

```python
from dataclasses import dataclass, field
from datetime import datetime
from typing import List, Optional

@dataclass
class ControllerState:
    process_id: str
    target: float
    tolerance: float
    safety_limit: float
    current_reading: float
    unit: str
    criticality_level: str      # NOMINAL|ADVISORY|ELEVATED|CRITICAL|EMERGENCY
    system_state: str           # RUNNING|SAFE|STOPPED
    actuator_output: float
    pid_integral: float
    pid_last_error: float
    active_sensor: str          # PRIMARY|BACKUP
    active_actuator: str        # PRIMARY|BACKUP
    stability_window_start: Optional[datetime]
    cycle_count: int
    start_time: datetime
    last_update: datetime

@dataclass
class SensorReading:
    sensor_id: str
    value: float
    unit: str
    timestamp: datetime
    status: str                 # OK|FAIL|STALE
    source: str                 # PRIMARY|BACKUP
```

```python
@dataclass
class ActuatorState:
    actuator_id: str
    output: float
    position: float
    status: str                 # OK|FAIL|OFFLINE
    mode: str                   # AUTO|MANUAL
    timestamp: datetime

@dataclass
class PersistenceRecord:
    version: str
    timestamp: datetime
    controller_state: ControllerState
    sensor_readings: List[SensorReading]
    actuator_states: List[ActuatorState]
    hmac_signature: str

@dataclass
class OperatorAlert:
    alert_id: str
    timestamp: datetime
    criticality_level: str
    process_id: str
    condition: str
    current_reading: float
    target: float
    safety_limit: float
    sensor_status: str
    actuator_status: str
    action_taken: str
```

```
    operator_action_required: bool
    authorisation_required: bool
```

---

## CONFIGURATION FILE

Location: `C:\Users\teach\Desktop\PythonCode\configs\industrial_controller.json`

```json
```

```
{
  "system": {
    "name": "Industrial Controller PSR",
    "version": "1.0",
    "cycle_interval_ms": 1000,
    "stability_window_minutes": 5
  },
  "process": {
    "id": "process_001",
    "name": "Reactor Temperature Control",
    "type": "TEMPERATURE",
    "target": 450.0,
    "tolerance": 2.0,
    "safety_limit": 480.0,
    "unit": "°C",
    "pid": {
      "kp": 1.0,
      "ki": 0.1,
      "kd": 0.05
    }
  },
  "sensors": {
    "primary": {
      "id": "RTD_001",
      "type": "RTD",
      "address": "192.168.1.100",
      "port": 5000,
      "stale_threshold_seconds": 30
    },
    "backup": {
      "id": "TC_001",
      "type": "IR_THERMOCOUPLE",
```

```
    "address": "192.168.1.101",
    "port": 5000,
    "stale_threshold_seconds": 30
  }
},
"actuators": {
  "primary": {
    "id": "HEATER_001",
    "type": "HEATER",
    "address": "192.168.1.200",
    "output_min": 0.0,
    "output_max": 85.0,
    "safe_position": 0.0,
    "signal_type": "4-20MA"
  },
  "backup": {
    "id": "HEATER_002",
    "type": "HEATER",
    "address": "192.168.1.201",
    "output_min": 0.0,
    "output_max": 85.0,
    "safe_position": 0.0,
    "signal_type": "4-20MA"
  }
},
"persistence": {
  "enabled": true,
  "path": "C:\\Users\\teach\\Desktop\\PythonCode\\data\\state.json",
  "interval_seconds": 10,
  "hmac_secret_key": "env:ICPSR_SECRET"
},
"reporting": {
```

```
    "scada_url": "env:SCADA_URL",
    "interval_seconds": 5,
    "use_tls": false,
    "queue_max_size": 1000
  },
  "resilience": {
    "watchdog_timeout_seconds": 30,
    "max_scada_retries": 3,
    "backoff_base_seconds": 2,
    "backoff_max_seconds": 60
  },
  "logging": {
    "path": "env:LOG_PATH",
    "level": "INFO"
  }
}
```

---

## ENVIRONMENT VARIABLES

| Variable | Purpose | Required | Verified Value |
|---|---|---|---|
| `ICPSR_SECRET` | HMAC signing key for state and alerts | Yes | SET (2026-04-19) |
| `SCADA_URL` | SCADA server URL | Yes | http://localhost:8443 (mock) |

| Variable | Purpose | Required | Verified Value |
|---|---|---|---|
| `CONFIG_PATH` | Configuration file path | Yes | C:\Users\teach\Desktop\PythonCode\configs\industri |
| `LOG_PATH` | Log file path | Yes | C:\Users\teach\Desktop\PythonCode\logs\icpsr.log |

## PROJECT STRUCTURE

```
C:\Users\teach\Desktop\PythonCode\
├── configs\
│   └── industrial_controller.json
├── data\
│   └── state.json              (created at runtime)
├── logs\
│   └── icpsr.log               (created at runtime)
├── industrial_controller\
│   ├── core\
│   │   ├── mission_loop.py     (REQ-001, REQ-025)
│   │   └── watchdog.py         (REQ-014)
│   ├── control\
│   │   └── pid_controller.py   (REQ-002, REQ-003, REQ-004)
│   ├── sensors\
│   │   └── adapter.py          (REQ-005, REQ-006)
│   ├── actuators\
│   │   └── adapter.py          (REQ-007, REQ-008)
│   ├── criticality\
│   │   └── classifier.py       (REQ-006, REQ-008, REQ-011, REQ-018, REQ-022,
REQ-023)
```

```
|   ├── persistence\
|   |   └── manager.py          (REQ-009, REQ-010, REQ-011, REQ-012, REQ-013)
|   ├── reporting\
|   |   └── scada_client.py     (REQ-015, REQ-016)
|   ├── alerts\
|   |   └── dispatcher.py       (REQ-017, REQ-018)
|   ├── safety\
|   |   └── safe_state.py       (REQ-019, REQ-020, REQ-024)
|   ├── auth\
|   |   └── gate.py             (REQ-021)
|   └── api\
|       └── server.py           (REST API endpoints)
├── tests\
|   ├── conftest.py
|   ├── test_mission_loop.py    (T-001, T-025, T-027)
|   ├── test_pid_controller.py  (T-002, T-003, T-004)
|   ├── test_sensors.py         (T-005, T-006)
|   ├── test_actuators.py       (T-007, T-008)
|   ├── test_persistence.py     (T-009, T-010, T-011, T-012, T-013)
|   ├── test_watchdog.py        (T-014)
|   ├── test_scada_client.py    (T-015, T-016)
|   ├── test_criticality.py     (T-017, T-018, T-022, T-023)
|   ├── test_safe_state.py      (T-019, T-020, T-021, T-024)
|   └── test_e2e_pipeline.py    (T-027)
├── industrial_controller.py    (CLI entry point)
└── requirements.txt
```

---

## DEPENDENCIES

```
# requirements.txt
```

```
# Environment: conda in_control
# Python: 3.13.1 (exact)

# Runtime
requests==2.32.3
cryptography==44.0.2
pyyaml==6.0.3
psutil==7.2.2

# Development and test
pytest==9.0.2
pytest-cov==7.0.0
```

---

## CLI INTERFACE

```powershell
```

```
# Start mission
python industrial_controller.py start

# Stop mission cleanly
python industrial_controller.py stop

# Force safe state immediately
python industrial_controller.py safe

# Resume from safe state (requires authorisation token)
python industrial_controller.py resume --token <auth_token>

# Current status
python industrial_controller.py status

# Show current configuration
python industrial_controller.py config
```

---

## PERFORMANCE REQUIREMENTS

| Metric | Requirement |
|---|---|
| Control cycle time | < 1 second end-to-end |
| Criticality classification | < 200ms per cycle |
| Sensor failover | < 1 cycle |
| State save time | < 2 seconds |

| Metric | Requirement |
|---|---|
| Startup and state restore | < 10 seconds |
| Safe state activation | < 3 seconds |
| Watchdog restart | < watchdog_timeout_seconds + 5 seconds |
| SCADA report dispatch | < 2 seconds per report |

## PROCESS TYPE REFERENCE

| Process | Target | Tolerance | Safety Limit | Unit |
|---|---|---|---|---|
| Polymerization | 180.0 | 2.0 | 200.0 | °C |
| Distillation | 78.0 | 1.0 | 85.0 | °C |
| Fermentation | 30.0 | 0.5 | 35.0 | °C |
| Calcination | 900.0 | 5.0 | 950.0 | °C |
| Reactor pressure | 150.0 | 5.0 | 180.0 | PSI |
| Water dosing flow | 12.0 | 0.5 | 15.0 | L/min |
| Tank level | 2.5 | 0.2 | 3.5 | m |
| Water treatment pH | 7.2 | 0.3 | 8.5 | pH |

*Problem first. Specification second. Facts before execution. Verification always.*

*SDPF Style 14 — Dynamic Criticality Extension Industrial Controller PSR v1 — Hamza Abdullah, Architect TVG Verified: 2026-04-19 — Environment: conda in_control — Python 3.13.1 — Windows 11*