TECHNICAL MANUAL

SL 3

USAREUR-AF

TM-30 — MAVEN SMART SYSTEM (MSS)

Foundation Course Manual

HEADQUARTERS
UNITED STATES ARMY EUROPE AND AFRICA
(USAREUR-AF)
Wiesbaden, Germany

26 MARCH 2026

DRAFT — UNOFFICIAL — NOT FOR OPERATIONAL USE

TM-30 — MAVEN SMART SYSTEM (MSS)

Forward: SL 3 qualifies data-adjacent specialists to build advanced no-code solutions on the Maven Smart System — complex Workshop applications, multi-source pipelines, well-designed ontology models, and advanced analytics — while operating within USAREUR-AF C2DAO governance requirements.
Prereqs: SL 1, Maven User; SL 2, Builder; Data Literacy Technical Reference (all required). *HQ USAREUR-AF · v1.0 · 2026 · DISTRIB: USG only · AUTH: C2DAO/UDRA v1.1*

TABLE OF CONTENTS

CHAPTER 1 — INTRODUCTION AND SCOPE

1-1. ADVANCED BUILDER MANUAL

1-1. This manual qualifies advanced builders who operate entirely through the MSS platform user interface. No code is required at the SL 3 level. All tasks in this manual are accomplished through Foundry's graphical tools: Workshop, Pipeline Builder, Ontology Manager, Contour, Quiver, and AIP Logic.

1-2. SL 3 is the advanced tier for data-adjacent specialists — personnel who work deeply with data but whose primary role is operational, analytical, or systems-oriented rather than software development. This includes 17-series and 25-series signal soldiers, S6/G6 staff, G2 analysts, Civil Affairs, and operational data analysts embedded in brigade and division staffs.

1-3. SL 4 covers code-level development (Python transforms, TypeScript Functions on Objects, OSDK). If your task requires writing code, reference SL 4. SL 3 stops at the UI boundary.

NOTE

All items above require a SL 4 developer. For SL 2 (no-code builder) capabilities, refer to TM-20, Chapter 1. SL 3 operates at the boundary between SL 2 no-code building and SL 4 code-based development. When in doubt whether a task is SL 2 or SL 3, refer to TM-20, Chapter 1-1 (No-Code Builder Manual) to assess scope before escalating.

1-4. Prerequisites. Before beginning SL 3 tasks, personnel must be qualified on: - SL 1 (Maven User): platform navigation, object search, consuming data products - SL 2 (Builder): basic Workshop, basic Pipeline Builder, basic Object Type configuration, basic Contour and Quiver - Data Literacy Technical Reference: data governance principles, Army data policy, command data authority

Before beginning SL 3 design work, confirm you can independently perform — without manual reference — all SL 1 operator tasks (Chapters 2-6) and all SL 2 builder tasks including Workshop application building (TM-20, Chapter 5), Ontology configuration (TM-20, Chapter 4), pipeline management (TM-20, Chapter 3), and branching/governance (TM-20, Chapter 7). If you cannot confidently perform any of these tasks without reference, complete the relevant TM before advancing.

1-2. WHAT SL 3 ADVANCES BEYOND SL 2

Capability Area	SL 2 Level	SL 3 Level
Workshop	Single-page apps, basic widgets	Multi-page apps, conditional logic, variable passing
Pipeline Builder	Single-source transforms, basic filters	Multi-source joins, aggregations, calculated columns
Ontology	Configure existing Object Types	Design Object Types, Link Types, and Actions from scratch
Contour	Basic charts and filters	Pivots, calculated columns, saved analysis views
Quiver	Single-object analysis	Multi-object dashboards, linked views, custom object sets
AIP Logic	Awareness	Configure and manage existing AI workflows
Governance	Follow naming conventions	Enforce them, review peers, coordinate with Data Stewards
Environment Mgmt	Aware of branching	Execute branching, review, and promotion workflows

NOTE

Advanced builders design solutions that SL 1 operators and SL 2 builders will use. Before designing at SL 3 level, understand the operator workflows from TM-10, Chapter 1 (Introduction and Overview) and TM-10, Chapter 4 (Using Workshop Applications). Your design decisions directly affect operator productivity and data quality at the operational level.

1-3. USAREUR-AF OPERATIONAL CONTEXT

1-5. USAREUR-AF is the Army Service Component Command (ASCC) to United States European Command (USEUCOM) and United States Africa Command (USAFRICOM). Advanced builders operating in this theater support land operations across the European and African AOR and integration with NATO Allied command structures. The data products built at SL 3 level feed readiness reporting, logistics visibility, intelligence products, and operational dashboards used by commanders at brigade through theater level.

1-6. Errors at SL 3 level have formation-wide impact. A broken Workshop application that misrepresents equipment readiness affects commanders' decisions. A poorly designed pipeline join that duplicates records inflates SITREP counts. A misconfigured Action that overwrites data without validation corrupts

the operational picture. Build with the same discipline you apply to any operational task.

1-7. Governance Authority. All MSS data products built within USAREUR-AF are governed by the Command and Control Data Authoritative Organization (C2DAO). Advanced builders are accountable to C2DAO for: - Naming convention compliance (datasets, Object Types, pipelines, applications) - Access control configuration — who can view, edit, and interact with your products - Data quality standards and issue reporting - Production promotion approval workflows

1-4. GOVERNING POLICY REFERENCES

1-8. The following policy documents govern advanced builder work. Builders must be familiar with the relevant sections.

Document	Authority	Key Provisions for Builders
Army CIO Memorandum (April 2024)	Army CIO/G-6	Data governance, data product ownership, access controls
AR 25-1, Army Information Technology (Jul 2019)	Army CIO	Statutory framework for Army data governance and IT management policy. NOTE: VAUTI (5 dimensions) from AR 25-1 is superseded by VAULTIS-A (8 dimensions, DDOF Playbook v2.2, Dec 2025). See Strategic Guidance below.
USAREUR-AF C2DAO Standards	C2DAO	Naming conventions, promotion gates, stewardship roles
CDA Portal (learn-data.armydev.com)	Army CDA	Training resources, design patterns, reference implementations

1-4a. Strategic Guidance

The following are strategic guidance documents — not doctrine — that inform MSS training design and operational context.

Document	Authority	Relevance
UDRA v1.1 (February 2025)	Army Enterprise	Unified Data Reference Architecture — domain alignment
DoD Data Strategy (October 2020)	OSD	Foundation for VAULTIS (7 dimensions); extended to VAULTIS-A (8 dimensions) by DDOF Playbook v2.2 (Dec 2025). Dimensions: Visible, Accessible, Understandable, Linked, Trusted, Interoperable, Secure, Auditable. 85% weighted avg = DDOF Phase 3 quality gate. Proponent: T2COM C2DAO / HQDA CIO/G-6 / SAIS-ADD.

Document	Authority	Relevance
Army Data Plan (2022)	Army CIO	11 strategic objectives for Army data transformation
Army Cloud Plan (2022)	Army CIO	Zero Trust, secure development, data-driven decisions
DoD Zero Trust Reference Architecture v2.0 (July 2022)	DoD CIO	Zero Trust architecture for data and application security
Army CIO Data Stewardship Memo (April 2024)	Army CIO	Chain of responsibility for data governance
NATO Data Strategy for the Alliance (Feb 2025)	NATO	Alliance-wide data governance mandate — governs coalition data sharing in EUCOM AOR
NATO Data Centric Reference Architecture v2 (2025)	NATO	Reference architecture for digital transformation — alignment target for coalition-interoperable ontology design
NATO Data Quality Framework for the Alliance (Aug 2025)	NATO	Quality governance and metrics — applicable to multi-national data products

1-9. CDA Portal. The Common Data Architecture (CDA) Portal at learn-data.armydev.com is the authoritative training and reference resource for Army data platform work. Advanced builders should consult the following CDA resources: - Object Type Cookbook v2 + Addendum A — authoritative Object Type design guidance - DDOF Playbook v2.2 — Defense Data Orchestration Framework lifecycle and quality gates - Doctrine-Driven Development framework — aligning ontology models to Army operational doctrine - ADP to JP to NATO Crosswalk — mapping Army, Joint, and NATO data constructs

1-10. DDOF Lifecycle. The Defense Data Orchestration Framework (DDOF) Playbook v2.2 (T2COM C2DAO / HQDA CIO/G-6 / SAIS-ADD, December 2025, CUI // FEDCON) governs the lifecycle of every data product built on MSS. Advanced builders must understand all six phases:

Phase	Name	Standard Duration	Gate Output
1	Problem Framing	3–5 days (max 10)	Approved requirement + SMART statement + ADC intake record
2	Data Provisioning	5–10 days (max 30)	Secured access to authoritative sources

Phase	Name	Standard Duration	Gate Output
3	Data Wrangling	7–12 days (max 20)	Clean data passing VAULTIS-A at 85%+ weighted average
4	Development	15–25 days (max 35)	Functional product with governance controls
5	Test & Evaluation	7–12 days (max 20)	Validated product, sponsor sign-off, 85%+ quality
6	Operations	Ongoing	Live product, ADC registration, monitoring operational

MVP Mandate: Per Secretary of the Army priorities, MVP delivery within 30 days. Extensions require C2DAO approval with documented justification.

Genesis Mission Alignment: DDOF implements three key directives: - **Decision Dominance** — Compress the OODA loop by delivering validated data products faster - **Bureaucracy Elimination** — Automate governance; pre-authorize access by mission role - **Accountability** — Explicit ownership at every phase with immutable audit trails

Quality Gates: Gates are enforced, not advisory. No progression without verified compliance. Products failing below 70% quality trigger remediate-or-retire. Products with no access in 90 days require FDM review; 180 days no access initiates retirement.

WARNING

Advanced builders who skip DDOF phases or bypass quality gates create technical debt that compounds at the enterprise level. Every product without proper Phase 1 framing, Phase 3 quality validation, or Phase 6 ADC registration degrades the command's data posture.

1-10a. DDOF Roles and Responsibilities

Every DDOF lifecycle phase requires clear role assignment. The following table defines the roles, echelons, and primary functions for data product development. (Source: DDOF Playbook v2.2, T2COM C2DAO, December 2025.)

Role	Echelon	Primary Function
Decision Maker (DM)	O-6+	Articulates requirements, approves final products
C2DAO	O-4/O-5	Governance oversight, quality assurance, gate approvals
Functional Data Manager (FDM)	O-3/O-4	Product ownership, lifecycle management
Data Engineer	Technical	Build and maintain data pipelines
Data Scientist/ORSA	Technical	Analytical modeling, algorithm development

Role	Echelon	Primary Function
Knowledge Manager	Staff	Documentation, training, organizational learning

Every data product must have a named FDM before entering Phase 2. The C2DAO validates role assignments at each quality gate. No product advances without an identified owner.

1-10b. SMART Criteria for Problem Framing (Phase 1)

Phase 1 Problem Framing requires a SMART statement before any development begins. Use this framework to validate the requirement is well-defined. (Source: DDOF Playbook v2.2, T2COM C2DAO, December 2025.)

Criterion	Definition	Example
Specific	What exactly do you need?	Equipment readiness by unit
Measurable	How will you know it's accurate?	Match GCSS-A within 2%
Achievable	Can it be built?	Platform and data exist
Relevant	Does it support a decision?	Tied to METL task
Time-bound	When must it be ready?	IOC by 15 JAN

A requirement that fails any SMART criterion is not ready for Phase 2. Return to the Decision Maker for refinement. Do not begin data provisioning against a vague requirement.

1-10c. Fail-Closed Enforcement

WARNING

Per Genesis Mission directives, data products must fail closed — denying access when authorization cannot be confirmed. Default deny. Explicit authorization required. All denials logged and audited. (Source: DDOF Playbook v2.2, T2COM C2DAO, December 2025.)

Fail-closed means: - If the authorization service is unavailable, deny access. - If a user's role cannot be verified, deny access. - If marking metadata is missing or malformed, deny access.

Do not implement fail-open patterns. Products that default to granting access on authorization failure will not pass Phase 5 Test & Evaluation.

1-10d. ADC Registration Requirements (Phase 6)

Phase 6 Operations requires registration in the Authoritative Data Catalog (ADC). Submit the following for every production data product. (Source: DDOF Playbook v2.2, T2COM C2DAO, December 2025.)

- Product name and description
- Owner (FDM) contact
- Classification and handling
- Data sources and lineage
- Refresh frequency
- Access method
- VAULTIS-A quality score

Products not registered in the ADC are not authorized for operational use. The C2DAO audits ADC completeness quarterly.

> **NOTE**
>
> ADC search is mandatory before any new development. Duplicate products waste resources and create confusion. If a similar product exists, modify it rather than creating new. Conduct an ADC search during Phase 1 and document the results in the intake record. (Source: DDOF Playbook v2.2, T2COM C2DAO, December 2025.)

1-10e. Data Operations in DDIL Environments

BLUF: Every MSS data product must function when network connectivity is denied, degraded, intermittent, or limited (DDIL). Products designed only for persistent connectivity fail when the operational environment degrades — which is when commanders need data most.

The Army Data Plan (2022) Strategic Objective 6 directs that data be "available at point of decision, at lower echelons, in DDIL environments." SO 10 further requires distributed decision-support capability across the range of military operations and changing command relationships. UDRA v1.1 Appendix E reinforces this: data products must be producible and consumable even with limited or no connectivity. ADP 6-0 warns that degraded communications require subordinates to possess enough shared understanding to act without continuous data flow.

Advanced builders address DDIL by designing degradation plans into every product from Phase 1. The following table defines the four DDIL conditions, their data impact, and required MSS mitigations:

DDIL Condition	Data Impact	MSS Mitigation
Denied	No network access	Pre-staged data packages, local compute
Degraded	Reduced bandwidth	Compressed data products, priority-based sync
Intermittent	Periodic connectivity	Store-and-forward queues, delta sync
Limited	Low bandwidth available	Text-only products, reduced refresh rates

Design every data product with these mitigations in mind. During Phase 1 Problem Framing, document which DDIL conditions the product must support. During Phase 3, validate that the product degrades gracefully under each applicable condition. During Phase 5 Test & Evaluation, test the product under simulated DDIL conditions before authorizing operational use.

WARNING

Builders who design products assuming persistent connectivity create single points of failure. Every MSS product must have a DDIL degradation plan.

(Sources: Army Data Plan (2022) SO 6, SO 10; UDRA v1.1 Appendix E; ADP 6-0.)

1-5. DESIGN PRINCIPLES FOR ADVANCED BUILDERS

BLUF: Advanced builders design systems that other people depend on. This section is not about Foundry features — it is about how to think before you use them. These principles apply across every SL 3 task.

1. Start with the command requirement, not the tool capability.

Foundry has many powerful tools. The most common advanced builder failure is selecting a tool first and then finding a use for it. Start here instead:

What does a commander or staff officer need to decide, and what information do they need to make that decision correctly?

Every piece of SL 3 work — an Ontology design, a multi-source pipeline, an AIP Logic workflow — should be traceable back to an operational requirement. If you cannot trace it back, you are building something that may not be needed.

2. Ontology design is domain modeling, not database configuration.

When you design Object Types and Link Types at SL 3 level, you are not configuring a database — you are modeling operational reality. The questions to ask:

- **What are the real-world entities?** (A Soldier, a vehicle, a unit, a maintenance event — these are nouns. Object Types are nouns.)
- **What are the real-world relationships?** (A Soldier *is assigned to* a unit. A vehicle *has* maintenance events. These are Links.)
- **What does a user need to do with these Objects?** (An S4 needs to update status. A G3 needs to approve a request. These are Actions — verbs.)

If you cannot describe your Ontology design in plain operational language without referencing Foundry concepts, the design is probably wrong. Test it: explain your Object Types to a non-technical officer. If they understand immediately, the model is sound. If they are confused, the model is unclear.

3. Every production change has downstream consequences.

At SL 3 level, the resources you modify are shared. An Object Type you change is used by Workshop applications, Contour analyses, and Quiver dashboards you may not even know exist.

Before modifying any shared production resource: 1. **Audit downstream dependencies** — Ontology Manager shows all downstream consumers of an Object Type. Run this audit before touching anything. 2. **Use a branch** — Never develop or test schema changes in production. Branch, build, test, then promote through the governance workflow. 3. **Communicate** — If your change will affect another team's application, notify them before you promote. This is not optional.

WARNING

Renaming a property on a production Object Type will break every downstream widget, filter, and computed column that references it by name. There is no automatic refactoring. Audit before you change.

4. Pipeline design — think about shape, not just content.

When building multi-source pipelines, the most common error is not getting wrong data — it is getting the right data in the wrong shape. Before configuring a join:

- **What is the grain of each source?** (One row per Soldier? One row per maintenance event? One row per day per unit?) Joining sources with different grains produces row multiplication. Know your grain before you join.
- **What is the intended grain of the output?** If the downstream Ontology expects one row per vehicle, your pipeline output must produce exactly one row per vehicle — no more, no fewer.
- **Where does deduplication happen?** If source systems can produce duplicate records, decide in the pipeline where to deduplicate — not in Workshop, not in Contour.

5. Your users are operators, not data scientists.

The applications and analyses you build at SL 3 level will be used by SL 1 operators — Soldiers and staff officers who are not data experts. Design for them:

- A dashboard that requires explanation to use has a design problem, not a user education problem.
- Use plain language for labels, filters, and column names. Do not expose internal dataset field names to users.
- If a user has to ask "what does this number mean?" — the answer should be visible in the application, not in a separate briefing.

NOTE

The CDA Portal (learn-data.armydev.com) Object Type Cookbook and DDOF Playbook contain reference patterns for Army-specific Ontology design. Consult these before designing novel Object Types from scratch.

PRODUCTION INCIDENT CASE STUDY — "The Renamed Property"

A SL 3 builder at V Corps was tasked with updating the `SoldierReadiness` Object Type to use clearer property names. One change seemed minor: renaming the property `unitStatus` to `unitOperationalStatus` — more descriptive, clearer in the UI.

The builder made the change on the main branch and published directly without a downstream impact assessment.

Within 30 minutes, three Workshop applications used by G1, G4, and the Brigade S3 stopped displaying readiness data. All three had been built to reference the property `unitStatus` by name. When the property was renamed, the widget bindings broke silently — the applications showed blank columns instead of errors, making the issue harder to detect.

V Corps G3 duty officer noticed the blank columns during an operational brief. The issue was escalated to C2DAO. Resolution required identifying all three broken applications, manually rebinding the property reference in each, and testing before repromoting. Total downtime: 4 hours.

What the builder should have done: 1. Performed a downstream impact assessment (Task 7-1) before any property rename. 2. Notified owners of all downstream applications of the pending change. 3. Made the change on a development branch. 4. Coordinated with downstream application owners to update their bindings before promotion. 5. Promoted only after all downstream applications were tested and confirmed functional.

Principle 3 applies: Every production change has downstream consequences. A 5-minute audit before the change would have prevented a 4-hour incident.

1-6. SL 3 SCOPE BOUNDARIES

SL 3 covers the full range of no-code, UI-based advanced building on MSS. Certain requirements exceed SL 3 scope and require SL 4 (software engineer) support.

Requirement	SL 3 Scope?	Action
Multi-page Workshop app with variable-driven navigation	YES	Build per Chapter 2
Multi-source join with grain management	YES	Build per Chapter 3
Complex Ontology with hierarchical Object Types	YES	Design per Chapter 4
AIP Logic workflow parameter configuration	YES	Configure per Chapter 6
Python or PySpark transforms	NO — SL 4	Refer to SL 4L
TypeScript Functions on Objects (FOO)	NO — SL 4	Refer to SL 4H/SL 4L

Requirement	SL 3 Scope?	Action
OSDK integration	NO — SL 4	Refer to SL 4L
Authoring new AIP Logic workflows	NO — SL 4	Refer to SL 4H
Configuring source connectors (new connector types)	NO — SL 4	Refer to SL 4L
Writing custom data quality checks in code	NO — SL 4	Refer to SL 4L

If a requirement is not in this table, apply the following rule: if it requires writing, editing, or debugging code in any language — it is SL 4 scope.

1-7. ADVANCEMENT FROM SL 3 — NEXT STEPS

SL 3 qualification enables advancement to all SL 4 tracks — both the six WFF functional tracks (SL 4A–F) and the eight specialist tracks (SL 4G–O). All SL 4 tracks require SL 3 as a hard prerequisite — SL 2 alone is not sufficient.

WFF Functional Tracks (SL 4A through SL 4F):

Track	Title	Audience
SL 4A	Intelligence	S2/G2 staff; intelligence analysts using MSS for WFF products
SL 4B	Fires	Fires staff; targeting analysts using MSS for fires products
SL 4C	Movement & Maneuver	M&M staff; using MSS for movement and maneuver products
SL 4D	Sustainment	Sustainment staff; using MSS for logistics and sustainment products
SL 4E	Protection	Protection staff; using MSS for protection products
SL 4F	Mission Command	MC staff; using MSS for mission command products

Specialist Tracks (SL 4G through SL 4O):

Track	Title	Specialty
SL 4G	ORSA	Operational research and systems analysis; quantitative modeling, statistical analysis, decision support
SL 4H	AI Engineer	AIP Logic workflow authoring, TypeScript Functions on Objects (FOO), Agent Studio
SL 4M	ML Engineer	Machine learning pipeline development, model integration, PySpark transforms

Track	Title	Specialty
SL 4J	Program Manager	Data product program management, delivery coordination, stakeholder engagement
SL 4K	Knowledge Manager	Organizational knowledge architecture, data product documentation, taxonomy management
SL 4L	Software Engineer	Python/PySpark transforms, TypeScript, OSDK integration, source connector configuration
SL 4N	UI/UX Designer	User interface design, user experience research, MSS application usability
SL 4O	Platform Engineer	Platform infrastructure, deployment pipelines, environment configuration

Each SL 4G–O track has a corresponding advanced track:

Advanced Track	Title	Prerequisite
SL 5G	Advanced ORSA	SL 4G (required)
SL 5H	Advanced AI Engineer	SL 4H (required)
SL 5M	Advanced ML Engineer	SL 4M (required)
SL 5J	Advanced Program Manager	SL 4J (required)
SL 5K	Advanced Knowledge Manager	SL 4K (required)
SL 5L	Advanced Software Engineer	SL 4L (required)
SL 5N	Advanced UI/UX Designer	SL 4N (required)
SL 5O	Advanced Platform Engineer	SL 4O (required)

Train-the-Trainer Track — available after SL 3:

T3-I (Instructor Certification) is a 5-day instructor pipeline course. Prereq: SL 3 Go + C2DAO Training OIC selection. Certifies personnel to deliver SL 2, SL 3, and SL 4 courses as primary instructor.

> **NOTE**
>
> There are no TM-50A through TM-50F tracks. Advanced-level training exists only for specialist tracks (G–O). WFF tracks (SL 4A–F) do not have a SL 5 continuation.

NOTE

Select the specialist track that aligns to your assigned duties and billet. If uncertain, consult the USAREUR-AF C2DAO training coordinator. Personnel are expected to complete one specialist track; concurrent enrollment in multiple SL 4 specialist tracks requires C2DAO approval.

CHAPTER 2 — ADVANCED WORKSHOP APPLICATIONS

BLUF: Advanced Workshop builders design multi-page applications with dynamic behavior — conditional visibility, variable-driven filtering, inter-page navigation, and complex layouts — that serve as the operational data interface for commanders and staff.

2-1. OVERVIEW OF ADVANCED WORKSHOP CAPABILITY

2-1. Workshop applications built at SL 3 level go beyond single-page dashboards. They serve as the primary operational interface for staff sections — a G2 intelligence dashboard with linked pages for threat assessment, unit tracking, and historical trends; an S4 readiness tracker that lets commanders drill from fleet overview to individual equipment status; a Civil Affairs civil-military operations board that filters by AOR, time period, and activity type simultaneously.

2-2. Advanced Workshop requires mastery of three interconnected concepts: variables, conditional logic, and multi-page navigation. These three elements, combined, allow you to build applications that behave like purpose-built operational software — without writing a line of code.

TASK 2-1: CONFIGURE APPLICATION VARIABLES AND STATE MANAGEMENT

TASK: Configure Workshop application variables, connect widgets to read and write those variables, and implement cascading variable chains to drive dynamic application behavior.

CONDITIONS: Builder has completed SL 1 and SL 2 and is SL 3 qualified. Access to a Workshop application in edit mode. SL 2 qualification confirmed.

STANDARDS: Builder configures application variables correctly, demonstrates variable passing between widgets and pages, and verifies dynamic behavior through test interactions before publishing. All variable names comply with C2DAO naming conventions.

EQUIPMENT: MSS platform access; Workshop edit permissions on the target application.

DURATION: 2–3 hours.

PROCEDURE:

Step 1 — Create application variables: 1. Open the application in Workshop edit mode. 2. Navigate to the Variables panel (left sidebar or top menu depending on Workshop version). 3. Select **Add Variable**. 4. Name the variable using the C2DAO naming convention: [domain]*[descriptor]*[type] — for example `equip_selected_unit_string` or `ops_date_filter_date`. 5. Set the variable type from the dropdown. 6. Set a default value appropriate to the expected initial state of the application. 7. Save the variable. It is now available to all widgets on all pages of the application.

> **NOTE**
>
> Variable names must be unique within the application and should be descriptive enough that another builder can understand the variable's purpose without opening a widget that uses it.

Variable types available in Workshop:

Variable Type	Use Case	Example
String	Text filter, selected category	Selected unit name, status code
Number	Numeric threshold, count	Days since last inspection
Boolean	Toggle, show/hide flag	Show classified records (true/false)
Date/DateTime	Time window selection	Report date, inspection DTG
Object	Selected ontology object	Selected vehicle record
Object Set	Filtered set of objects	All vehicles in a brigade with status RED

Step 2 — Connect a dropdown widget to a variable (write): 1. Select the dropdown widget in edit mode. 2. Open widget **Configuration**. 3. Under **On Selection**, set **Write to Variable** and select the target variable. 4. Define the source of dropdown options: static list, property of an Object Type, or a dataset column. 5. Verify that the selected value matches the variable type (string to string, etc.). 6. Save.

Step 3 — Connect a table widget to a variable (read/filter): 1. Select the table widget in edit mode. 2. Open widget **Configuration**. 3. Under **Filters**, add a new filter condition. 4. Set the filter property (e.g., unit_name), set the operator (equals), and set the value source to **Variable**, then select the target variable. 5. Optionally configure behavior when the variable is empty: show all records, show no records, or show a default filtered set. 6. Save and test by interacting with the dropdown in preview mode.

> **CAUTION**
>
> Configuring a table to show all records when the filter variable is empty can return extremely large datasets on production Object Types. Set a default value for variables used as table filters, or configure the table to show no records until a selection is made.

Step 4 — Configure cascading (chained) dropdowns: 1. Create two variables: `ops_selected_division_string` and `ops_selected_brigade_string`. 2. Configure the first dropdown (Division) to write to `ops_selected_division_string`. 3. Configure the second dropdown (Brigade) to: a. Source its options from the Brigade Object Type. b. Apply a pre-filter on the options list using `ops_selected_division_string` (filter to brigades where parent_division equals the variable). c. Write the user's brigade selection to `ops_selected_brigade_string`. 4. Configure downstream tables and charts to filter on `ops_selected_brigade_string`. 5. Test the complete chain: select a division, verify brigade dropdown updates, select a brigade, verify tables filter correctly.

NOTE

Always consider what should happen when a parent variable changes. If a user selects a new division, the brigade variable retains its previous value until the user makes a new brigade selection. If the previous brigade does not belong to the new division, downstream widgets will show no results. Add a reset mechanism (a button that clears child variables) or configure the brigade dropdown to auto-clear when the division variable changes.

TASK 2-2: IMPLEMENT CONDITIONAL LOGIC AND VISIBILITY RULES

TASK: Configure Workshop widgets and panels to show or hide based on application state, and apply conditional formatting within widgets to indicate status values.

CONDITIONS: Builder has completed SL 1 and SL 2 and is SL 3 qualified. Existing multi-widget Workshop application available in edit mode. SL 2 qualification confirmed.

STANDARDS: Builder correctly applies conditional visibility rules so that widgets and panels display or hide based on application state, without any visible layout breaks in the published application.

EQUIPMENT: MSS platform access; Workshop edit permissions on the target application.

DURATION: 1–2 hours.

NOTE

Conditional layouts determine which panels or pages operators (SL 1) see based on their role or selected data state. Test your conditional layout logic against the operator workflows in TM-10, Chapter 4. Do not hide information from operators without a security or role-based justification. Refer to TM-10, Task 4-1 (Orient to a Command-Level Application) and Task 4-3 (Apply Filters to a Dashboard) to validate layout behavior from the operator's perspective.

PROCEDURE:

Step 1 — Configure conditional widget visibility: 1. Select the widget or panel to be conditionally shown/hidden. 2. Open widget **Configuration** then **Visibility**. 3. Select **Conditional**. 4. Define the condition: Variable, Operator, Value. Example: show this panel only when `ops_selected_unit_string` is not empty. 5. Use compound conditions (AND/OR) for more complex logic. Example: show only when `ops_selected_unit_string` is not empty AND `ops_view_mode_string` equals detail. 6. Save and test in preview mode.

Common conditional visibility patterns for operational applications:

Pattern	Implementation
Detail panel on selection	Show detail widget when selected-object variable is not empty
Role-based view	Show admin controls when user-role variable equals steward
Empty state message	Show "Select a unit to view status" when filter variable is empty
Progressive disclosure	Show advanced filters when show_advanced_boolean is true
Alert banner	Show warning panel when data_staleness_hours variable exceeds threshold

Step 2 — Configure conditional formatting within a table (traffic light status): 1. Select the table widget. 2. Open **Column Configuration** for the status column. 3. Enable **Conditional Formatting**. 4. Add rules: - Value equals NON-MISSION CAPABLE — Red background or red icon - Value equals PARTIALLY MISSION CAPABLE — Amber background or amber icon - Value equals FULLY MISSION CAPABLE — Green background or green icon 5. Set a fallback format for unexpected values (gray or default). 6. Save and preview.

> **NOTE**
>
> Use the Army standard readiness categories (FMC, PMC, NMC) consistently. Do not invent status labels — align to the authoritative data source definitions.

Step 3 — Configure conditional widget content (beyond visibility): 1. Beyond visibility, certain widgets support conditional content — displaying different values, colors, or icons based on data conditions. This operates within the widget itself, not the variable system. 2. In Workshop table widgets, configure row-level conditional formatting via column properties. 3. For metric tiles, configure icon or color changes based on threshold values. 4. For text widgets, use variable interpolation to display dynamic values. 5. Test all conditional content paths in preview mode before publishing.

TASK 2-3: DESIGN A MULTI-PAGE WORKSHOP APPLICATION

TASK: Design and build a multi-page Workshop application with logical page structure, cross-page variable passing, and complete navigation tested end-to-end.

CONDITIONS: Builder has completed SL 1 and SL 2 and is SL 3 qualified. Workshop application with more than one logical section or audience required. Edit permissions on the application.

STANDARDS: Builder creates a logical page structure with clear navigation, passes variables across pages correctly, and verifies that the published application loads each page without error.

EQUIPMENT: MSS platform access; Workshop edit permissions; design sketch of page layout completed before building.

DURATION: 4–8 hours depending on application complexity.

> **NOTE**
>
> Decision framework — single-page vs. multi-page: If your application serves a single user role or a single operational workflow, design single-page (SL 2 scope — refer to TM-20, Chapter 5-3). If your application serves multiple user roles simultaneously (e.g., G3 operations, G4 logistics, G6 data), or integrates multiple workflows into a unified interface, design multi-page (SL 3 scope). Multi-page application navigation is what operators experience via TM-10, Task 4-1 (Orient to a Command-Level Application). Test your navigation design against that task.

PROCEDURE:

Step 1 — Design page structure before building: 1. On paper or whiteboard, define: - What is the primary question each page answers? - Who uses each page and what action do they take on it? - What data flows from one page to the next (what variables carry across)? - What is the navigation pattern (menu, breadcrumb, button-driven drill-down)? 2. Review the standard page patterns for USAREUR-AF operational applications:

Page Type	Purpose	Common Widgets
Overview	Command summary, fleet or force status at a glance	Metric tiles, status chart, map
Drill-Down	Detail for a selected unit, asset, or event	Object detail panel, linked table
Filter/Search	User-driven exploration of a large dataset	Search bar, multi-filter panel, results table
Admin	Data entry, action execution, record updates	Forms, Action buttons, validation messages

Page Type	Purpose	Common Widgets
Governance	Data quality flags, issue reports, stewardship views	Issue queue table, action buttons

1. Do not begin building until the page design is agreed upon with the data steward and key stakeholders.

Step 2 — Create the multi-page structure in Workshop: 1. In the application editor, open **Page Manager** (accessible from top navigation or sidebar). 2. Add pages using **Add Page** — name each page clearly using the pattern [Number]_[PageName] (e.g., 01_Overview, 02_UnitDetail, 03_Readiness). 3. Set the landing page (the page users see on first load). 4. Add a **Navigation** widget or a **Button** widget to each page that links to other pages. 5. For sidebar navigation: configure the navigation widget with the list of page names and their target pages. 6. For button-driven drill-down: configure each button's **On Click** action to navigate to the target page. 7. Test navigation in preview mode: verify all page links resolve, no broken navigation exists.

Step 3 — Implement selection-driven cross-page navigation: 1. On the Overview page, add an Object Set or Table widget that displays units. 2. Configure the widget's **On Row Click** or **On Object Select** to write the selected object to a variable (e.g., `ops_selected_unit_object`). 3. Add a button or auto-navigate action that transitions to the UnitDetail page on selection. 4. On the UnitDetail page, configure all widgets to filter or resolve using the `ops_selected_unit_object` variable. 5. Add a **Back** button that navigates to the Overview page. 6. Test the complete flow end-to-end.

> **NOTE**
>
> When navigating back to the Overview page, the selection variable retains the previous value unless you explicitly clear it. This is usually desirable — users can see their previous selection highlighted. If you need a clean state, add a button action that clears the variable before navigating back.

Step 4 — Verify the complete application before publishing: 1. Test every navigation path in preview mode. 2. Test every variable interaction on every page. 3. Verify conditional visibility rules function on all pages. 4. Test with production-representative data volumes. 5. Have a second builder or the data steward test independently before publishing.

TASK 2-4: CONFIGURE COMPLEX WIDGETS (TABLES, DYNAMIC OBJECT SETS, MAPS)

TASK: Configure advanced Workshop widgets — including computed table columns, multi-condition dynamic object sets, and layered map displays — for operational application use.

CONDITIONS: Builder has completed SL 1 and SL 2 and is SL 3 qualified. Target Object Types have appropriate geometry, date, and relationship properties configured. Edit permissions on the application.

STANDARDS: Builder configures complex widgets with correct data bindings, tests all interactive behaviors with production-representative data, and verifies load performance before publishing.

EQUIPMENT: MSS platform access; Workshop edit permissions; target Object Types published with required properties.

DURATION: 2–4 hours per widget type.

PROCEDURE:

Step 1 — Configure a table with computed columns: 1. Open the table widget configuration. 2. Under **Columns**, select **Add Computed Column**. 3. Name the column clearly: `days_since_inspection`, `readiness_pct`, `shortfall_count`. 4. Build the formula using the formula editor. Common operations available: - Arithmetic: `property_a + property_b`, or `property_a / property_b * 100` - Date math: `today() - last_inspection_date` (returns days as integer) - Conditional: `if(status == "NMC", 0, 1)` - String: `concat(unit_name, " - ", equipment_type)` 5. Set the column data type (number, string, date) to match the formula output. 6. Apply conditional formatting to the computed column if useful (e.g., red when days > 30). 7. Save and verify values are correct against known records.

> **CAUTION**
>
> Computed columns in Workshop tables are calculated at display time for the current page of results. They do not create a permanent dataset. If you need the computed value available in pipelines, Actions, or other applications, compute it in Pipeline Builder instead and store the result in a dataset.

Step 2 — Configure a multi-condition dynamic object set: 1. Add or select an Object Set widget. 2. Open **Filter Configuration**. 3. Add the first filter condition: property, operator, value (or variable). 4. Select **Add Condition** and choose AND or OR. 5. Add the second condition. Repeat for additional conditions. 6. To use a variable as a filter value: set the value source to **Variable** and select the target variable. 7. Configure the **Empty Variable Behavior**: show all, show none, or show a default set. 8. Set the **Sort Order** to control display priority (e.g., sort by `readiness_status` ascending so NMC records appear first). 9. Configure visible properties to show only columns relevant to the current page. 10. Save and test.

Step 3 — Configure an advanced map widget: 1. Add a map widget to the application page. 2. Under **Layers**, add the primary Object Type as the first layer. Set marker style (icon, color) based on a status property using conditional formatting. 3. Add additional Object Types as separate layers with distinct marker styles. 4. Under **Pop-up Configuration**, define which properties appear when a user clicks a map marker. Include operationally relevant fields (unit, status, last update DTG). 5. Under **Variable Integration**, configure map selection to write the selected object to an application variable, driving detail

panels on the same page. 6. Under **Basemap**, select an appropriate operational basemap. Avoid consumer map services for sensitive operational overlays. 7. Test the map with production-representative data. Verify load time is acceptable.

> **NOTE**
>
> Map widgets require that Object Types have geometry or coordinate properties configured in the Ontology. If the Object Type lacks location data, the map widget will not display objects. Coordinate with the Data Steward to verify location property availability before designing a map-dependent application.

2-2. DESIGNING FOR OPERATIONAL TEMPO AND DATA CURRENCY

Advanced builders designing applications for time-sensitive operational use must account for data freshness and potential degraded conditions.

Data Currency Markings in Workshop

Use visual cues to indicate data freshness in applications consumed by commanders: - Add a text widget to every dashboard header displaying: "Data as of: [Last Updated timestamp from backing dataset]" - For datasets with known refresh schedules, add a conditional color indicator: green if within scheduled window, yellow if delayed >1 hour, red if delayed >4 hours - Mark data from unit-submitted forms as "Unofficial" until validated by the data steward

Default Filtering for Large Datasets

Workshop applications that query large Object Types (>50,000 objects) without default filters will load slowly under operational conditions. Configure a required default filter on every production dashboard (e.g., default to the user's unit, current date, or active records only). Label the filter "Required — select a unit" so operators know they must filter before results appear.

> **CAUTION**
>
> An application configured to show all records with no default filter on a large Object Type can return results that take 20–30 seconds to load during peak usage. In a time-critical brief, this is operationally unacceptable. Always test load time with production-representative data volumes before publishing.

Communicating Data Uncertainty

If a data source is known to have reliability issues (e.g., a feed that occasionally delays or drops records), add an informational banner to the application: "Data source: [Name]. Known refresh issues may cause delay. Verify with [Source System] if data appears stale."

Do not present uncertain data as authoritative. It is better to acknowledge uncertainty than to have a commander brief wrong numbers.

NOTE

Additional Workshop capability (self-study) — SL 3 class time covers multi-page navigation, variable passing, and conditional logic. The Workshop Scenarios feature (what-if analysis, saved scenario states, preloaded application states) is covered in four Palantir Developers reference videos: *Workshop | Creating What If Analyses with Scenarios*, *Workshop | Saving your What If Analyses*, *Workshop | Loading and Applying Scenarios*, and *Workshop | How to Preload States in Foundry Workshop Applications*. These are UI-based capabilities within SL 3 scope that builders can apply to decision-support applications requiring scenario comparison.

2-5. KAIROS TIMELINE INTEGRATION

What Kairos Is

Kairos is Palantir Foundry's timeline and planning visualization tool. It displays time-sequenced events, tasks, and activities plotted against Ontology objects — enabling builders to surface operational timelines, planning horizons, and activity schedules directly within Workshop applications.

Kairos vs. Gantt charts: A Gantt chart is a static document — it represents a plan at a point in time and must be manually updated. Kairos is Ontology-driven: data flows live from Object Types through defined date properties into the timeline display, and updates in real time as the underlying data changes. A maintenance schedule managed as Object data in MSS appears in Kairos as an automatically current timeline — no manual updates required.

Use Kairos when: - The operational question involves sequences, durations, or scheduling (e.g., "What maintenance events are occurring this week across the brigade?") - A commander or staff officer needs to visualize activities against a time axis - Multiple parallel activity streams need to be displayed side by side (e.g., units, equipment categories, or staff functions as swim lanes)

TASK 2-5: CONFIGURE A KAIROS TIMELINE WIDGET IN WORKSHOP

TASK: Add and configure a Kairos timeline widget in a Workshop application, connecting it to an existing Object Type with date properties, and configure swim lanes and groupings to produce an operationally meaningful timeline view.

CONDITIONS: Builder has completed SL 1 and SL 2 and is SL 3 qualified. A Workshop application exists in edit mode. At least one Object Type is published with at minimum two date or datetime properties (a start date and an end date or duration). Builder is working on a development branch.

STANDARDS: Builder adds a Kairos widget to the application, connects it to the correct Object Type, configures start and end date properties, defines at least one swim lane grouping, and verifies the timeline displays correctly with production-representative data before publishing.

EQUIPMENT: MSS platform access; Workshop edit permissions on the target application; Object Type with date properties; development branch active.

DURATION: 2–4 hours.

PROCEDURE:

Step 1 — Add the Kairos widget to the application:

1. Open the Workshop application in edit mode (on your development branch).
2. In the Widget Library (left panel), locate the **Kairos** or **Timeline** widget. Drag it onto the application canvas.
3. Resize and position the widget. Timeline widgets typically benefit from a wide, horizontal layout.

Step 2 — Configure the data source:

1. In the widget properties panel (right), under **Data Source**, select the Object Type that contains the events or activities to display.
2. Under **Start Date**, select the property that defines when each event or activity begins.
3. Under **End Date** or **Duration**, select the property that defines when each event ends (or the duration). If objects have only a start date and no end date, configure the Kairos widget to display point events (no duration bar, just a marker on the timeline).
4. Under **Label**, select the property that will identify each event on the timeline (e.g., event name, equipment identifier, unit name).

Step 3 — Configure swim lanes and groupings:

1. Under **Grouping** or **Swim Lanes**, select the property to use as the swim lane key. Each unique value of this property becomes a separate horizontal lane on the timeline. Examples:
2. Group by `battalion_name` — one lane per battalion
3. Group by `equipment_category` — one lane per equipment type
4. Group by `staff_function` — one lane per warfighting function
5. Set the swim lane label to display clearly (the property value that names each lane).
6. If the timeline has many swim lanes, configure the **Collapse** option to allow users to expand and collapse individual lanes.

Step 4 — Configure visual options:

1. Under **Color Coding**, configure event bar colors based on a property value (e.g., readiness status — green for FMC, amber for PMC, red for NMC). This requires the Object Type to have a status property with defined values.
2. Configure **Filtering** to connect the Kairos widget to an application variable (date range selector, unit filter), so users can narrow the timeline to a relevant window or formation.
3. Configure the **Default Time Window** — the initial date range displayed when the application loads. Set this to the most operationally relevant window (e.g., current week, current month).

Step 5 — Test and verify:

1. In preview mode, verify that events appear correctly on the timeline.
2. Test the time range navigation (scrolling or date range selection).
3. Test swim lane expansion and collapse.
4. Test color coding against known records to verify colors match status values.
5. Test any connected variables (date range filter, unit filter) to verify the timeline updates when filter selections change.

> **NOTE**
>
> Kairos timeline performance degrades with very large object sets (thousands of events visible simultaneously). Configure a date range filter as a required input so users select a time window before the timeline loads. Do not configure Kairos to display all events with no date filter on large Object Types.

TTPs for refining Kairos visualizations: - Keep swim lane count manageable — more than 20 swim lanes simultaneously visible creates visual clutter. Use a unit/group filter to reduce the number of visible lanes. - Use color coding purposefully — limit to 3–5 distinct colors. More colors than that are difficult to distinguish in a command brief environment. - Ensure event labels are concise — long event names truncate in the timeline bar. Use abbreviations where operationally standard. - Align time window defaults to your audience's planning horizon: operational staff think in days, logistics staff may need weeks, training managers may need months.

2-6. TARGET WORKBENCH

What Target Workbench Is

Target Workbench is a Foundry-native application supporting structured targeting workflows. It integrates Ontology objects into a kill-chain and targeting cycle workflow, providing intelligence and fires staff with a structured interface for managing targeting packages, tracking target status through the targeting cycle, and coordinating engagement decisions.

NOTE

Target Workbench is primarily relevant to intelligence (G2/S2) and fires (fires support officer, FA) staff functions. Builders in sustainment, personnel, signal, or other functional areas should understand what Target Workbench is and that it exists — but it is not a tool you will build against unless you are supporting an intelligence or fires function. Operational use is covered in SL 4A (Intelligence WFF) and SL 4B (Fires WFF).

Builder responsibility: Advanced builders supporting intelligence or fires functions are responsible for ensuring that the Object Types and properties in the Ontology are correctly structured to support Target Workbench display. Target Workbench reads from Object Types — if those Object Types are incorrectly configured (wrong property names, missing required fields, broken Link Types), Target Workbench will display incomplete or incorrect targeting data.

TASK 2-6: NAVIGATE AND ORIENT TO TARGET WORKBENCH

TASK: Navigate to Target Workbench, orient to its interface and data model, and identify the Ontology Object Types that feed into it.

CONDITIONS: Builder has completed SL 1 and SL 2 and is SL 3 qualified. Builder is supporting an intelligence or fires function. Target Workbench has been deployed in the command's MSS environment and access has been granted by the Data Steward.

STANDARDS: Builder navigates to Target Workbench, identifies the targeting workflow stages displayed, traces the connection between displayed data and the underlying Ontology Object Types, and can explain to a SL 4 developer or Data Steward which Object Types need to be correctly structured for Target Workbench to display complete data.

EQUIPMENT: MSS platform access; Target Workbench access; Ontology Manager access.

DURATION: 1–2 hours orientation.

PROCEDURE:

1. Use the platform Search bar to locate Target Workbench, or navigate to it via Compass if it has been organized into your command's application folder.
2. Open Target Workbench and orient to the interface:
3. Review the targeting stages or phases displayed (the targeting cycle steps: detect, identify, decide, execute, assess, or equivalent command-specific framework)
4. Identify how targets are organized and displayed (by priority, by status, by geographic area)
5. Review any target records visible and note which properties are displayed for each target
6. Navigate to Ontology Manager. Identify the Object Types that feed Target Workbench by searching for Object Types with names related to targeting, intelligence objects, or named areas of interest. Your SL

4 developer or unit data steward can provide the specific Object Type names if they are not immediately identifiable.

7. Open one of the identified Object Types and compare its configured properties to the fields displayed in Target Workbench. Confirm that the Object Type's properties include all fields Target Workbench requires.
8. Identify any properties that appear missing, incorrectly named, or incorrectly typed — these are the builder's responsibility to fix before Target Workbench displays complete data.
9. Document findings and coordinate with the Data Steward and SL 4 developer on any required Ontology fixes.

NOTE

Do not modify Target Workbench application configuration directly. Target Workbench is a managed Palantir application — your role as a SL 3 builder is to ensure the underlying Ontology data is correctly structured. Application-level Target Workbench configuration is SL 4H/SL 4A/SL 4B scope.

CHAPTER 3 — ADVANCED PIPELINE BUILDER

BLUF: Advanced Pipeline Builder work at SL 3 level involves joining multiple source datasets, applying complex transformations and aggregations, and producing analysis-ready outputs — all through the visual pipeline interface without writing code.

3-1. PIPELINE BUILDER REVIEW AND SL 3 SCOPE

NOTE

TM-20, Chapter 3 covered single-source ingestion pipelines with basic transformations. SL 3 advances to multi-source joins, complex business logic transforms, error handling that prevents silent failures, and monitoring strategy. Before designing a SL 3 pipeline, confirm the requirement genuinely exceeds SL 2 pipeline capabilities (TM-20, Chapter 3-1). If the requirement can be met with a single-source pipeline and basic transforms, build it at SL 2 level and do not escalate unnecessarily.

3-1. SL 2 covered single-source Pipeline Builder work: reading a dataset, applying column selection, basic filters, and renaming. SL 3 advances to multi-source operations — joins, unions, aggregations, and derived columns using Pipeline Builder's visual transform library.

3-2. Pipeline Builder represents each transformation as a visual node connected by data flow edges. At SL 3 level, pipelines will include multiple input branches that merge, transform, and flow into one or more outputs. Readability of the pipeline graph is itself a quality standard — other builders must be able to read your pipeline diagram and understand what it does.

TASK 3-1: PERFORM A MULTI-SOURCE JOIN

TASK: Configure a Pipeline Builder join of two or more source datasets on a shared key field, select the correct join type, handle NULL values from unmatched rows, and verify output row counts.

CONDITIONS: Builder has completed SL 1 and SL 2 and is SL 3 qualified. Two or more source datasets with a common key field available. Pipeline Builder access confirmed. Source dataset schemas reviewed and documented before beginning.

STANDARDS: Builder completes a join correctly, verifies row counts before and after to confirm join logic, identifies and handles NULL values from unmatched rows, and documents the join key in the pipeline node description.

EQUIPMENT: MSS platform access; Pipeline Builder permissions; documented source dataset schemas.

DURATION: 2–4 hours including testing.

PROCEDURE:

Step 1 — Select join type:

Join Type	Returns	Use When
Inner Join	Only rows that match in both datasets	You only want records that exist in both sources
Left Join	All rows from left dataset; NULLs where no right match	You want all records from the primary dataset, with optional enrichment from secondary
Right Join	All rows from right dataset; NULLs where no left match	Same as left, with datasets reversed
Full Outer Join	All rows from both datasets; NULLs on both sides where no match	You need a complete picture including unmatched records on either side

USAREUR-AF Example: Joining a vehicle fleet dataset (left) to a maintenance record dataset (right). Use a Left Join — you want every vehicle, whether or not it has a maintenance record. Vehicles with no maintenance record will show NULL in maintenance fields. An Inner Join would silently drop vehicles with no maintenance history, underreporting fleet size.

Step 2 — Configure the join in Pipeline Builder: 1. Open Pipeline Builder and create or open a pipeline. 2. Add two **Dataset** input nodes — one for each source. Label each node clearly (right-click then rename): `src_vehicle_fleet`, `src_maintenance_records`. 3. Add a **Join** transform node from the transform library. 4. Connect the left dataset edge to the Join node's left input port. 5. Connect the right dataset edge to the Join node's right input port. 6. Open the Join node configuration. 7. Select the join type (Inner, Left, Right, or Full Outer). 8. Define the join key: select the matching column from the left dataset and the matching column from the right dataset. These must be the same data type (string-to-string, integer-to-integer). 9. If joining on multiple keys (composite key), add additional key pairs using **Add Key Pair**. 10. Review the output schema preview — verify expected columns are present. 11. Add a **Row Count** check (using a branch to a Count node) to verify the output row count is within expected range before proceeding.

CAUTION

Joining on a column that contains NULL values in either dataset will cause those rows to not match. If the join key may contain NULLs, add a **Filter** node upstream to remove or handle NULL keys before the join.

NOTE

After a join, column names from both datasets are merged into one schema. If both datasets have a column with the same name (other than the join key), Pipeline Builder will prefix them. Rename these columns immediately after the join using a **Rename Columns** node for readability.

Step 3 — Configure a multi-source union (when combining same-schema sources): 1. Add two or more **Dataset** input nodes with compatible schemas. 2. Add a **Union** transform node. 3. Connect all input datasets to the Union node. 4. Open Union configuration and verify column mapping — Pipeline Builder will attempt to auto-map columns by name. Review the mapping and correct any mismatches. 5. Enable **Deduplicate** if the same record may appear in multiple sources. 6. Preview output and verify row count equals the expected sum of input rows (minus duplicates if deduplication is enabled).

Step 4 — Validate join output: 1. Compare the output row count to the expected count given the join type. 2. Sample output records and verify joined values are correct. 3. Check for unexpected NULL columns that indicate join key mismatches. 4. Document the join key, join type, and rationale in the Join node description field.

TASK 3-2: CONFIGURE DATASET AGGREGATIONS

TASK: Configure a Pipeline Builder Group By aggregation to summarize a dataset, add calculated columns from aggregated values, and verify output against expected results.

CONDITIONS: Builder has completed SL 1 and SL 2 and is SL 3 qualified. Source dataset with repeating rows requiring summarization available. Clear definition of the grouping key and the aggregation metrics required before beginning.

STANDARDS: Builder configures aggregation grouping and metrics correctly, verifies output row count is less than input (confirming rollup occurred), and names output columns descriptively.

EQUIPMENT: MSS platform access; Pipeline Builder permissions.

DURATION: 1–3 hours.

PROCEDURE:

Step 1 — Configure a Group By aggregation: 1. Add a **Group By** transform node downstream of your data source or join. 2. Open Group By configuration. 3. Under **Group By Keys**, add the columns that define each unique group. Example: `unit_name` and `equipment_type` — one output row per unique combination. 4. Under **Aggregations**, add the metrics: - COUNT(*) — total records in the group - SUM(column) — total value - AVG(column) — mean value - MAX(column) / MIN(column) — range -

COUNT_DISTINCT(column) — unique value count 5. Name each aggregated output column clearly: `total_vehicles`, `avg_readiness_pct`, `max_days_overdue`. 6. Preview output and verify the number of rows equals the expected number of unique groups.

USAREUR-AF Example — Readiness Rollup by Unit: Group By keys: `battalion_name`, `equipment_category` Aggregations: COUNT(*) to `vehicle_count`, SUM(`is_fmc`) to `fmc_count`, AVG(`readiness_pct`) to `avg_readiness_pct` Output: one row per battalion per equipment category showing fleet count and average readiness.

Step 2 — Add calculated columns from aggregated values: 1. Add a **Calculated Column** node downstream of the Group By node. 2. Open configuration and select **Add Column**. 3. Name the new column: `readiness_rate_pct`. 4. Build the formula using the no-code formula editor: - Arithmetic: `fmc_count / vehicle_count * 100` - Conditional: `if(readiness_rate_pct >= 90, "GREEN", if(readiness_rate_pct >= 70, "AMBER", "RED"))` - Date math: `date_diff(today(), last_inspection_date, "days")` 5. Set the output data type. 6. Preview and verify values against known records.

> **NOTE**
>
> Calculated columns in Pipeline Builder produce a permanent column in the output dataset — unlike Workshop computed columns, which are display-only. Use Pipeline Builder calculated columns when the value will be used in the Ontology, in Actions, or in other downstream pipelines.

Step 3 — Verify aggregation output: 1. Confirm the output row count is less than the input row count (rollup occurred). 2. Manually calculate the expected value for one known group and compare to pipeline output. 3. Verify column names are descriptive and follow C2DAO naming conventions. 4. Document the aggregation logic in the Group By node description.

TASK 3-3: IMPLEMENT AN ADVANCED TRANSFORMATION PATTERN

TASK: Implement a multi-source union with deduplication to combine same-schema records from multiple subordinate unit feeds into a single analysis-ready dataset.

CONDITIONS: Builder has completed SL 1 and SL 2 and is SL 3 qualified. Two or more source datasets with compatible schemas available. Data Steward has confirmed deduplication key field and logic.

STANDARDS: Builder configures the union with correct schema mapping, implements deduplication correctly, verifies output represents the expected unique record count, and documents the deduplication logic.

EQUIPMENT: MSS platform access; Pipeline Builder permissions; documented source schemas.

DURATION: 2–4 hours.

PROCEDURE:

Step 1 — Configure the union: 1. Add a **Dataset** input node for each source (e.g., SITREP records from multiple subordinate units). 2. Label each input node clearly: `src_1cd_sitreps`, `src_3id_sitreps`, `src_173abn_sitreps`. 3. Add a **Union** transform node and connect all input nodes. 4. Open Union configuration and verify column mapping is correct across all sources. 5. Note any schema differences (extra columns, renamed columns) and resolve before continuing.

Step 2 — Implement deduplication: 1. After the Union node, add a **Deduplicate** transform node. 2. Define the deduplication key — the column(s) that uniquely identify a record (e.g., `sitrep_id`, `report_dtg` plus `unit_uic`). 3. If multiple rows have the same key, define the row selection rule: keep the most recent (by a timestamp column), keep the first, or aggregate. 4. Preview output and verify the count is at or below the expected unique record count.

Step 3 — Handle NULL values: 1. After deduplication, apply NULL handling appropriate to the data: - Use **Filter** to drop rows with NULL in required key fields. - Use **Calculated Column** with `coalesce(column, "UNKNOWN")` to replace NULLs in categorical fields. - Flag records with NULLs in important fields using a boolean calculated column for downstream data quality review.

Step 4 — Configure pivot transforms (when required for output shape): 1. If downstream consumers need wide-format data, add a **Pivot** transform. 2. Set the **Row Key**: the column that identifies each unique row in the output (`unit_name`). 3. Set the **Pivot Column**: the column whose unique values become new column headers (`equipment_type`). 4. Set the **Value Column**: the column whose values fill the pivoted cells (`vehicle_count`). 5. Set the **Aggregation**: SUM, AVG, MAX, etc. 6. Preview output — verify columns match the expected pivot structure.

CAUTION

Pivot creates one output column for each unique value in the Pivot Column. If the Pivot Column has high cardinality (hundreds of unique values), the output will have hundreds of columns, which degrades performance and readability. Limit pivot operations to low-cardinality categorical fields (fewer than 20 unique values).

3-2. PIPELINE NAMING AND DOCUMENTATION STANDARDS

NOTE

When a SL 3 pipeline fails, the downstream impact is broad. Operators (TM-10, Task 5-1, View and Read a Dataset) see stale or missing data. Workshop applications fed by the pipeline display errors. Complex data products may serve dozens of operators or downstream pipelines. Refer to TM-10, Chapter 7-1 (Common Problems and Solutions) to understand the operator experience of a pipeline failure, then design your monitoring and alerting to detect failures before operators report them.

3-3. Every pipeline produced at SL 3 level must conform to C2DAO naming and documentation standards. A pipeline that cannot be identified, understood, or maintained by another builder is a governance deficiency.

3-4. **Pipeline naming convention:** `[domain]_[source-description]_[output-description]_[version]` Example: `log_vehicle-fleet-maintenance-join_readiness-rollup_v1`

3-5. **Required pipeline documentation:** - **Pipeline Description:** What does this pipeline produce? What question does it answer? - **Source Datasets:** List all input datasets with dataset names and owners. - **Output Dataset:** Name, intended consumer (Workshop app, Ontology Object Type, Contour view). - **Join Keys:** Document all join keys and join types with rationale. - **Refresh Cadence:** How often should this pipeline run? On schedule or on-demand? - **Data Steward:** Who is responsible for this pipeline?

3-6. **Dataset partitioning awareness.** Advanced builders do not configure partitioning in the UI — that is a SL 4 code-level task — but must design with it in mind. A partitioned dataset stores data in segments organized by a partition key (typically a date column). Apply date filters on partition keys as early as possible in the pipeline (immediately after the source node, before any joins). Ask the Data Steward whether a source dataset is partitioned and on what key. A missing or late-applied partition filter is the most common cause of slow pipelines.

NOTE

Pipeline health monitoring (self-study) — Once pipelines are in production, monitoring their health is a SL 3 responsibility. The Pipeline Monitoring capability is covered in two Palantir Developers reference videos: *Pipeline Monitoring | How to Start Monitoring Data Health in Palantir Foundry* and *Pipeline Monitoring | How to Monitor Health Across a Pipeline in Palantir Foundry*. Similarly, the Schedules feature (configuration, metrics, and trigger management) is covered in three videos: *Schedules | Creation, Configuration, and Execution*, *Schedules | Management, Metrics, and Triggers*, and *Schedules | Separating Data Ownership within a Pipeline*. These tools are UI-based and within SL 3 scope.

CHAPTER 4 — ONTOLOGY DESIGN THROUGH THE UI

BLUF: Advanced builders design Object Types, Link Types, and Actions through the Ontology Manager UI — making deliberate design decisions that affect all downstream applications, analytics, and workflows that consume the Ontology. Good design prevents downstream pain; poor design forces expensive rework.

4-1. ONTOLOGY AS OPERATIONAL DATA MODEL

4-1. The Ontology is the shared data model of the platform. It defines what things exist in your operational environment (Object Types), how they relate to each other (Link Types), and what users can do to them (Actions). Every Workshop application, Quiver analysis, Contour chart, and AIP Logic workflow that touches operational data does so through the Ontology.

4-2. Ontology design decisions are not easily reversed. Changing a property name, deleting a Link Type, or restructuring an Object Type after other teams have built applications against it causes those applications to break. Design carefully. Review with Data Stewards before publishing.

4-3. SL 3 scope: This chapter covers designing Object Types, Link Types, and Actions through the Ontology Manager graphical interface. Writing TypeScript Functions on Objects (FOO) or code-level ontology configuration is SL 4 scope.

NOTE

SL 2 Ontology configuration (TM-20, Chapter 4) is limited to: (1) simple Object Types with straightforward properties; (2) one-to-one and one-to-many Link Types without junction complexity; (3) single-step Actions with direct form-to-field mapping. If a design requires multi-step Actions, conditional routing, derived properties with complex logic, or many-to-many Link Types beyond a simple junction, it is SL 3 scope. When assessing complexity, use TM-20, Chapter 4-2 (Ontology Manager Interface Overview) as the boundary reference.

TASK 4-1: DESIGN AND CREATE AN OBJECT TYPE FROM A COMMAND REQUIREMENT

TASK: Starting from a documented command requirement, design and create an Object Type in Ontology Manager with a correctly defined primary key, well-named properties, appropriate visibility settings, and full documentation.

CONDITIONS: Builder has completed SL 1 and SL 2 and is SL 3 qualified. Identified operational entity to model. Source data schema documented and available. Coordination with Data Steward on naming and property standards completed before beginning.

STANDARDS: Builder creates an Object Type with a correctly defined primary key, well-named properties with correct data types, appropriate visibility settings, and documentation in the description field. Object Type passes Data Steward review before publication to production.

EQUIPMENT: MSS platform access; Ontology Manager edit permissions; documented source data schema; Data Steward contact confirmed.

DURATION: 2–4 hours including Data Steward coordination.

PROCEDURE:

Step 1 — Answer the five design questions before opening Ontology Manager: 1. **What is the primary key?** Every object instance must be uniquely identifiable. What field guarantees uniqueness? (UIC for units, NSN plus serial for equipment, DODAAC plus document number for transactions.) 2. **What properties describe this entity?** List the fields that will be stored on each object instance. Include what is operationally meaningful and what downstream consumers will actually use. 3. **What does this entity link to?** What other Object Types does it relate to? 4. **Who owns this data?** Who is the Data Steward? Who authorizes changes? 5. **What is the source dataset?** What Pipeline Builder output backs this Object Type?

Step 2 — Create the Object Type: 1. Navigate to Ontology Manager in the platform. 2. Select **Create Object Type**. 3. Set the Object Type **Name**: use PascalCase, singular noun. C2DAO convention: [Domain][EntityName] — example: `LogVehicle`, `OpsUnit`, `MedFacility`. 4. Set the **API Name** (auto-generated from name — verify it follows convention). 5. Set the **Description**: one to three sentences describing the entity, its source, and its operational purpose. 6. Set the **Primary Key Property**: select the property that uniquely identifies each object. This is typically the UIC, serial number, equipment ID, or document number. 7. Add **Properties** for each attribute: - Name: camelCase, descriptive (`unitName`, `equipmentStatus`, `lastInspectionDate`) - Data type: String, Integer, Double, Boolean, Timestamp, Date — match the source data type - Mark required properties as required - Add a description to each property that is not self-evident 8. Set **Visibility**: who can see this Object Type? Apply the least-permissive setting that meets operational requirements. 9. Review configuration. Request Data Steward review before publishing.

NOTE

Do not create an Object Type for every dataset you have. Object Types are for entities that need to be tracked, searched, acted upon, or related to other entities over time. Reference data and lookup tables that are only consumed within pipelines do not need Object Types.

Step 3 — Verify property design quality:

NOTE

When designing properties for an Object Type, consider how operators (SL 1) will understand and use them. Refer to TM-10, Task 5-3 (Use Quiver to Explore Ontology Objects) and Task 4-3 (Apply Filters to a Dashboard) to see how operators interact with your property names and values. Use clear operational terminology. Avoid technical abbreviations operators will not recognize.

Avoid these common property design errors:

Error	Impact	Correct Practice
Using String for a numeric value	Cannot sort numerically, cannot aggregate	Use Integer or Double for numeric fields
Using String for a date	Cannot perform date math, cannot filter by range	Use Date or Timestamp
Over-generic property names	Other builders cannot use the Object Type without documentation	Use descriptive names matching operational terminology
Including PII without authorization	Data governance violation	Coordinate with Data Steward and legal before including PII
Omitting description on non-obvious properties	Downstream builders don't know what the field means	Add a description to every property where the name alone is insufficient

Status properties: if a status field has a defined set of valid values (FMC, PMC, NMC; GREEN, AMBER, RED; ACTIVE, INACTIVE), document those values in the property description. This enables downstream builders to configure accurate Workshop filters and Contour charts.

TASK 4-2: CREATE A LINK TYPE AND CONFIGURE CARDINALITY

TASK: Create a Link Type between two existing Object Types, configure correct cardinality and directionality, define link labels, and submit for Data Steward review.

CONDITIONS: Builder has completed SL 1 and SL 2 and is SL 3 qualified. Two existing Object Types with a defined operational relationship. Coordination with Data Steward completed before beginning.

STANDARDS: Builder creates a Link Type with the correct directionality, cardinality, and naming. Link Type is documented and passes Data Steward review.

EQUIPMENT: MSS platform access; Ontology Manager edit permissions; both target Object Types already published.

DURATION: 1–2 hours.

PROCEDURE:

Step 1 — Define the Link Type design:

Decision	Options	Guidance
Directionality	One-directional or bi-directional	Most operational links are bi-directional (a vehicle belongs to a unit; the unit has vehicles)
Cardinality	One-to-one, one-to-many, many-to-many	Reflect the actual operational reality — a vehicle has one primary unit; a unit has many vehicles
Link name	Should read naturally in both directions	LogVehicle to OpsUnit: forward = "assigned to", reverse = "has assigned"

Step 2 — Create the Link Type: 1. In Ontology Manager, navigate to **Link Types** and select **Create Link Type**. 2. Set the **Name**: use descriptive language that reflects the relationship. Convention: `[ObjectTypeA]_[RelationshipVerb]_[ObjectTypeB]` Example: `LogVehicle_assignedTo_OpsUnit` 3. Set the **Source Object Type** (the "from" side of the relationship). 4. Set the **Target Object Type** (the "to" side of the relationship). 5. Set cardinality: select one-to-one, one-to-many, or many-to-many. 6. Set **Link Labels**: the label used to describe the relationship in each direction. Source to Target: "is assigned to"; Target to Source: "has assigned vehicle" 7. Define the **Link Key**: the shared property that connects the two object types (e.g., `unit_uic` present on both `LogVehicle` and `OpsUnit`). 8. Add a description explaining the operational meaning of this link. 9. Submit for Data Steward review before publishing.

WARNING

Creating a Link Type on the wrong property pair creates incorrect relationships that appear valid in the UI but produce misleading results in Quiver, Workshop, and AIP Logic. Verify the link key produces the expected relationships by previewing a sample of linked objects before publishing.

TASK 4-3: DESIGN A MULTI-STEP ACTION WITH APPROVAL ROUTING

TASK: Design and configure a multi-step Action in Ontology Manager with conditional routing and approval chain logic for a workflow requiring command authority sign-off.

CONDITIONS: Builder has completed SL 1 and SL 2 and is SL 3 qualified. Identified workflow where users need to create, update, or delete ontology object data through the platform UI. Data Steward authorization to create write-back capability obtained before beginning.

STANDARDS: Builder creates an Action with correct parameter definitions, appropriate validation rules, clear user-facing labels, and appropriate authorization controls. Action is tested end-to-end in a non-production environment before promotion.

EQUIPMENT: MSS platform access; Ontology Manager edit permissions; Data Steward authorization documented.

DURATION: 3–5 hours including testing.

> **NOTE**
>
> Distinguish SL 2 Actions from SL 3 Actions: SL 2 Actions are single-step — operator fills form, field is updated. SL 3 Actions support multi-step workflows, conditional routing, and approval chains. If an Action requires: (1) sequential submission steps; (2) conditional field visibility; (3) multi-record writes; (4) command authority approval — it is SL 3 scope. If the workflow can be expressed as a single form-to-field write, hand it back to a SL 2 builder.

PROCEDURE:

Step 1 — Define the Action design before building: - What is the operational workflow this Action supports? - Who initiates the Action? Who approves it? Who is notified? - What data writes occur at each step? - What validation is required to prevent bad data entry?

Common USAREUR-AF Action use cases: - S4 NCO updates equipment status from NMC to PMC after partial repair - G2 analyst flags an intelligence report as reviewed and adds a confidence rating - Unit administrator creates a new equipment record when a vehicle arrives in theater - Data Steward marks a data quality issue as resolved - Operations officer logs a significant activity event with location and description

Step 2 — Create the Action: 1. Navigate to **Actions** in Ontology Manager and select **Create Action**. 2. Set the **Action Name**: use an imperative verb phrase that describes what the action does from the user's perspective. Example: Update Equipment Status, Log Maintenance Event, Flag Data Quality Issue. 3. Set the **Action Type**: Create Object, Modify Object, Delete Object, or Batch Modify. 4. Select the **Target Object Type**. 5. Define **Parameters** — the fields the user will fill in when executing the action: - Parameter name (label shown to the user) - Data type (String, Integer, Boolean, Date, Object reference) -

Required or optional - Default value if applicable - Validation rules (see Step 3) 6. Map parameters to **Object Properties**: which parameter value writes to which property on the object. 7. Set **Authorization**: who can execute this action? Apply least-privilege — limit to the role that genuinely needs write access. 8. Set the **Confirmation Message**: what does the user see before the action executes? Make it specific enough that the user understands what will change. 9. Set the **Success Message**: what does the user see after the action executes? 10. Test the action in a non-production environment.

Step 3 — Configure validation rules:

Validation rules prevent bad data from entering the Ontology through Actions. Every Action parameter that accepts user input must have at least one validation rule.

Validation Type	Use Case	Example
Required field	Prevent empty submissions	Status update requires a status value
Allowed values list	Limit input to valid options	Status must be FMC, PMC, or NMC
Minimum / Maximum	Numeric range enforcement	Readiness percentage must be 0-100
Date range	Prevent impossible dates	Event date cannot be in the future
Length limit	String field size control	Notes field maximum 500 characters
Pattern match	Format enforcement	UIC must match the defined UIC format

Configure a clear, user-facing validation error message for each rule so users understand what correction is needed.

NOTE

Validation rules in Actions are the last line of defense before data enters the Ontology. Treat them as critically as input validation at any other system boundary. A missing validation rule is a data quality risk that will require remediation after the fact.

Step 4 — Configure approval routing (for Actions requiring command authority): 1. If the Action requires an approval step before the write executes, configure **Approval Routing** in the Action configuration. 2. Define the approver role or specific users who can approve. 3. Configure the pending state: what the initiator sees while waiting for approval. 4. Configure the approval notification: what the approver sees and what information they need to make the decision. 5. Configure outcomes: approve (execute the write) or reject (Action does not execute; initiator is notified). 6. Test the full approval chain end-to-end in a non-production branch.

TASK 4-4: DEFINE AN INTERFACE ON AN OBJECT TYPE

CONDITIONS: Builder has a set of Object Types that share common properties (e.g., multiple equipment types that all have status, location, and assigned_unit) and needs to define a reusable contract across them.

STANDARDS: Builder creates an Interface that defines shared properties, applies it to two or more Object Types, and verifies that Workshop components using the Interface work across all implementing types.

EQUIPMENT: Ontology Manager access, at least two Object Types with overlapping properties, development branch.

PROCEDURE:

1. Identify the shared properties across your Object Types. List them: property name, data type, and purpose.
2. In **Ontology Manager**, navigate to the **Interfaces** section.
3. Click **Create Interface**. Name it descriptively using C2DAO naming standards (e.g., `StatusTrackable`, `GeoLocatable`, `UnitAssignable`).
4. Add each shared property to the Interface definition. For each property, specify the name and data type — these must match exactly across all implementing Object Types.
5. Navigate to each Object Type that should implement this Interface. In the Object Type configuration, add the Interface under **Implemented Interfaces**.
6. Map each Interface property to the corresponding property on the Object Type. If a required property does not exist on the Object Type, add it first (Task 4-2).
7. **Verify:** Open a Workshop application that uses one of the implementing Object Types. Confirm the Interface-based widgets (tables, charts, forms) render correctly for each implementing type.
8. Test with at least two different Object Types to confirm the Interface contract holds.

NOTE

Interfaces enable builders to create Workshop components that work across multiple Object Types without duplicating widget configuration. A table configured for the `StatusTrackable` interface will display any Object Type that implements it — units, vehicles, personnel — without reconfiguration.

CAUTION

Changing an Interface definition after multiple Object Types implement it can break downstream Workshop applications. Treat Interface changes as schema changes — assess downstream impact using lineage (Chapter 7, Task 7-1) before modifying.

CHAPTER 5 — ADVANCED ANALYTICS: CONTOUR AND QUIVER

BLUF: Advanced analytics at SL 3 level moves beyond basic charts into complex aggregations, cross-object analysis, and saved analytical views that become persistent operational intelligence products.

5-1. ADVANCED CONTOUR OVERVIEW

NOTE

Contour at SL 2 level (TM-20, Chapter 6) supports basic filtering, sorting, and simple aggregations. Operators use Contour via TM-10, Task 5-2 (Use Contour for No-Code Analysis). SL 3 Contour adds the formula editor for calculated columns, complex multi-table aggregations, and pivot analysis. Before designing at SL 3 level, confirm the analysis requirement exceeds SL 2 Contour capabilities (TM-20, Chapter 6-2). If SL 2 Contour can handle the requirement, build there first.

TASK 5-1: BUILD AN ADVANCED CONTOUR ANALYSIS WITH PIVOT AND FORMULA LOGIC

TASK: Build a Contour analysis using multi-level Group By aggregation, calculated columns with conditional formulas, and pivot analysis, and save the result as a shareable operational analysis product.

CONDITIONS: Builder has completed SL 1 and SL 2 and is SL 3 qualified. Published dataset or Object Type with sufficient data for analysis available. Contour access confirmed. Analysis requirement documented and agreed upon with the requestor before beginning.

STANDARDS: Builder produces analysis with correct aggregations, meaningful calculated columns, saved views configured for reuse, and shared appropriately with the intended audience.

EQUIPMENT: MSS platform access; Contour access; published dataset or Object Type.

DURATION: 2–4 hours.

PROCEDURE:

Step 1 — Configure a multi-level Group By in Contour: 1. Open Contour and select your dataset or Object Type. 2. In the analysis pane, add a **Group By** operation. 3. Add the primary grouping key (e.g., `division_name`). 4. Add a secondary grouping key (e.g., `equipment_category`). 5. Add aggregation

metrics: COUNT, SUM, AVG, MAX, MIN. 6. Run the analysis and verify output structure. 7. Add additional **Calculated Column** operations to derive ratios or composite metrics from the aggregated values.

Step 2 — Add conditional calculated columns: 1. Conditional aggregation is achieved by combining a calculated column with an aggregation: a. Add a **Calculated Column**: `is_nmc = if(status == "NMC", 1, 0)`. b. Add an **Aggregation**: SUM(`is_nmc`) to produce `nmc_count`. c. Add another **Calculated Column**: `nmc_rate = nmc_count / total_count * 100`. 2. Contour calculated columns are ephemeral — they exist in the analysis session and saved views, but do not persist in the underlying dataset. This is ideal for exploratory analysis and reporting without modifying source data. 3. Name columns descriptively. Add a comment in the view description explaining non-obvious formulas.

Step 3 — Create a pivot analysis: 1. Add a **Pivot** transform in the analysis pane. 2. Set the **Row Keys** (left-side group): `battalion_name`. 3. Set the **Pivot Column** (column headers): `equipment_category`. 4. Set the **Value**: the metric to display in cells (`readiness_rate_pct`). 5. Set the **Aggregation**: how to combine multiple values per cell (AVG, SUM, etc.). 6. Run and verify the pivot structure matches the expected layout. 7. Add a **Total** row or column if the analysis requires summary margins.

Step 4 — Select the appropriate chart type:

Chart Type	Best For	Avoid When
Bar chart	Comparing values across categories	More than approximately 15 categories
Line chart	Trends over time	Non-time-series data
Scatter plot	Correlation between two numeric values	Categorical data
Heat map	Intensity across two categorical dimensions	Fewer than 5 categories per axis
Pie/donut	Part-to-whole relationships	More than 6 slices
Table	Precise values, multiple metrics per row	Data is better communicated visually
Map	Geographic distribution	Data has no meaningful geographic component

Step 5 — Save and share the analysis view: 1. Configure the analysis in Contour: filters, groupings, calculated columns, chart type. 2. Select **Save View** (or **Save As** for a new view from an existing one). 3. Name the view using the C2DAO naming convention: `[domain]_[product-name]_[frequency-or-audience]` Example: `log_readiness-by-unit_weekly`, `ops_sitrep-trend_monthly`. 4. Add a description: what is this analysis? Who uses it? What question does it answer? 5. Set sharing: share with specific users, groups, or make available to all authorized users. 6. Add to the relevant Workshop application if this analysis is an embedded product. 7. For recurring reporting products, configure dynamic date filters — "last 7 days," "current month" — so the view always reflects the current period without manual reconfiguration.

TASK 5-2: BUILD A MULTI-OBJECT QUIVER DASHBOARD

TASK: Build a Quiver dashboard that displays analysis across multiple Object Types simultaneously, with views linked so that object selection in one view filters related views.

CONDITIONS: Builder has completed SL 1 and SL 2 and is SL 3 qualified. Published Ontology with defined Object Types and Link Types available. Quiver access confirmed.

STANDARDS: Builder creates multi-object analyses, configures linked views, defines custom object sets, and produces shareable analytical modules that link correctly across object types.

EQUIPMENT: MSS platform access; Quiver access; Ontology with at least two Object Types and a Link Type connecting them.

DURATION: 2–4 hours.

PROCEDURE:

Step 1 — Create a compound filter object set: 1. Open Quiver and select an Object Type. 2. In the filter panel, add the first filter condition. 3. Add additional conditions using AND/OR operators. 4. Use property-level filters: equals, not equals, greater than, less than, contains, is NULL. 5. Use object relationship filters — filter based on properties of a linked Object Type. 6. Name and save the object set for reuse.

> **NOTE**
>
> Saved object sets in Quiver can be embedded in Workshop applications as the data source for Object Set widgets. This creates a single definition of a critical operational set (e.g., "All NMC vehicles in 21st TSC") that is used consistently across multiple applications.

Step 2 — Build the multi-object dashboard layout: 1. Open a new Quiver dashboard or layout. 2. Add an **Object Set View** for the primary Object Type (e.g., Units). 3. Add a second **Object Set View** for a related Object Type (e.g., Vehicles assigned to units). 4. Configure **Linked Views**: in the Vehicle view, set the filter source to the Unit view's selection — when a user selects a unit in the Unit view, the Vehicle view automatically filters to that unit's vehicles. 5. Add metrics and charts to each view as needed. 6. Add a **Summary Panel** at the top showing aggregate counts across both object types. 7. Save and share the dashboard.

Step 3 — Create custom metrics: 1. In the Object Type's Quiver configuration, select **Add Custom Metric**. 2. Name the metric clearly: `nmc_vehicle_count`, `avg_readiness_pct`. 3. Define the formula using the metric builder: - For a simple calculated value: use arithmetic on existing properties. - For an aggregation over linked objects: select the Link Type, the target Object Type, and the aggregation

(COUNT, SUM, AVG) on a target property. 4. Set the display format: number, percentage, date. 5. Preview the metric on sample objects. 6. Save. The metric is now available in all Quiver analyses on this Object Type.

NOTE

Extended Quiver capability (self-study) — The Quiver tasks above cover multi-object dashboards and linked views. Four Palantir Developers reference videos go deeper on specific capabilities: *Quiver | How to Navigate the Dependency Graph and Expand your Analysis* (tracing upstream dependencies within a Quiver analysis), *Quiver | How to Use Parameters in Your Analysis* (parameter-driven filtering), *Quiver | How to Perform Ad-Hoc Aggregations*, and *Quiver | Calculating KPIs for Time Series Data in Palantir Foundry*. The time-series KPI video is particularly relevant for readiness trend analysis use cases.

CHAPTER 6 — AIP LOGIC CONFIGURATION

BLUF: At SL 3 level, builders configure and manage AIP Logic workflows and their operational parameters through the UI — activating, tuning, and monitoring AI-assisted processes without authoring underlying model logic.

6-0. AIP LANDSCAPE AT SL 3 LEVEL

Before working with any AIP tool, understand the full AIP toolset and the scope each tool represents at SL 3 level. Palantir's AI Platform (AIP) includes several distinct capabilities that serve different purposes and require different qualification levels to configure.

AIP Logic is the platform's rule-based and AI-assisted workflow automation layer. It enables builders to connect AI reasoning to Ontology data and configure automated processes that surface insights, flag conditions, or trigger actions based on data state. SL 3 builders configure existing AIP Logic workflows — adjusting parameters, connecting data sources, enabling natural language query. Full workflow authoring is SL 4 scope. Tasks 6-1 and 6-2 (below) cover the SL 3 AIP Logic configuration scope in full.

AIP Analyst is a natural language interface that allows users to query Ontology objects and datasets using plain English. A user can type "Which vehicles in 3rd ABCT have been deadlined for more than 30 days?" and AIP Analyst returns structured results from the Ontology — without requiring the user to write a filter or run a Contour analysis. AIP Analyst is distinct from AIP Logic: Logic is automated (runs on schedule or trigger without user prompting); Analyst is interactive (the user queries on demand). Builder role at SL 3: ensure the Object Types and properties backing the data are correctly configured with clear names, accurate data types, and meaningful descriptions — these directly affect the quality of AIP Analyst query results. No dedicated task is required for AIP Analyst at SL 3 level; orientation is sufficient.

AIP Agent Studio is Foundry's environment for building AI agents that can reason over Ontology data and take actions on behalf of users. Agent Studio is addressed in Task 6-3. SL 3 covers basic agent creation — connecting Object Types, writing output instructions, and testing responses. Advanced agent development — custom tool configuration, agentic action logic (agents that write back to objects or trigger workflows), production deployment, and multi-agent orchestration — is SL 4H (AI Engineer) and SL 4G (ORSA) scope.

AI FDE (AI-Driven Feature Development Environment) is Palantir's platform capability for code-assisted AI product development. It is addressed in section 6-4 as an awareness item. Hands-on AI FDE development is SL 4H scope.

NOTE

At SL 3 level, you are a consumer and configurator of AIP capabilities — not an author. Your primary contribution to AIP quality is well-designed Ontology structure: clear property names, accurate data types, complete descriptions, and correct Link Types. A poorly structured Ontology produces poor AIP outputs regardless of how well the AI model is configured. Build the data layer correctly first; AI quality follows from data quality.

6-1. SL 3 VS. SL 4 SCOPE IN AIP LOGIC

SL 3 builders CONFIGURE existing AIP Logic workflows — they do not author them from scratch.

Activity	Scope	Who Does It
Adjust workflow input parameters (thresholds, filter values, schedule)	SL 3	Advanced builder
Connect a workflow to a new data source (via UI)	SL 3	Advanced builder
Modify plain-language prompt text for operational context	SL 3	Advanced builder
Enable/disable natural language query on an Object Type	SL 3	Advanced builder
Author a new AIP Logic workflow from scratch	SL 4	AI engineer (SL 4H)
Write TypeScript functions or modify workflow logic code	SL 4	AI engineer (SL 4H)
Integrate AIP Logic with external systems	SL 4	AI engineer (SL 4H)

WARNING

AIP Logic configurations that make autonomous decisions affecting command intent — such as flagging soldiers as non-deployable, modifying readiness records, or triggering escalation actions — require command authorization and legal review before activation. Do not activate production AIP Logic workflows that affect command decision-making without written authorization from your data steward and unit commander.

NOTE

If you are unsure whether your configuration crosses into SL 4 territory, stop and consult your C2DAO data engineer. The cost of escalating unnecessarily is low. The cost of authoring workflows outside your qualification is high.

6-1a. Q1 2026 AIP Logic Feature Updates

The following capabilities were added to AIP Logic in Q1 2026 and are within SL 3 configuration scope:

Branch selection for evals run configuration. AIP Logic now supports selecting a specific branch when configuring evaluation (evals) runs. Builders can run evals against a development branch before promoting to production — verifying that workflow parameter changes produce correct outputs in the branch environment without affecting live data. When configuring an evals run, select the target branch under **Run Configuration > Branch**. Always run evals against your development branch first; do not run untested evals against the production branch.

Debugging panel for AIP Logic configuration. A dedicated debugging panel is now available within the AIP Logic configuration interface. The panel displays real-time execution traces, input/output inspection for each workflow step, and error diagnostics — enabling builders to identify configuration issues without escalating to a SL 4 developer for every troubleshooting event. Access the debugging panel from the **Debug** tab within any AIP Logic workflow configuration view. Use it to: - Inspect the inputs and outputs of each workflow step during a test execution - Identify which step in the workflow produced an unexpected result - View error messages and stack traces for failed executions - Verify that data source connections are resolving correctly

NOTE

The debugging panel is a diagnostic tool — it does not allow modification of workflow logic. If the panel reveals a logic error in the workflow itself (not a configuration error), escalate to the SL 4 developer. Configuration errors (wrong data source, incorrect parameter value, bad filter) are SL 3 scope and should be corrected directly.

6-2. AIP LOGIC OVERVIEW

NOTE

AIP Logic is SL 3 only. SL 2 builders do not configure AI workflows. SL 1 operators use AIP Logic workflows that SL 3 builders design — see TM-10, Task 4-8A (Read and Respond to an AIP Logic Alert) and Task 4-8B (Interact with an AIP Agent) for the operator's perspective. Operators must review and validate AI outputs before acting on them (emphasized in TM-10, section 4-8). Design your AIP Logic workflows so outputs are easy for operators to validate quickly. Workflows that produce outputs requiring extensive operator review are operationally inefficient.

6-1. AIP Logic is the AI workflow layer of the platform. It enables AI-assisted analysis, automated reasoning, and natural language interfaces to operational data. At SL 3 level, builders do not author AIP Logic workflows — that is SL 4 scope. SL 3 builders: - Configure the operational parameters of existing

AIP Logic workflows - Connect workflows to appropriate data sources and Object Types - Activate and deactivate workflows in production - Monitor workflow health and output quality - Report configuration issues to SL 4-level developers

6-2. **CAUTION:** AIP Logic workflows that consume operational data can produce outputs that appear authoritative. Before activating an AIP Logic workflow in production: - Verify the workflow has been reviewed and approved by the responsible Data Steward - Verify users will see clear labeling indicating outputs are AI-generated - Confirm the command authority understands the scope and limitations of AI-assisted analysis - Ensure the workflow has a human-in-the-loop review step for any decision-relevant output

---

TASK 6-1: CONFIGURE AIP LOGIC WORKFLOW PARAMETERS

TASK: Configure operational parameters of an existing AIP Logic workflow — including data source connections, prompt text, scheduling, and confidence thresholds — and test with representative sample data before activating in production.

CONDITIONS: Builder has completed SL 1 and SL 2 and is SL 3 qualified. Existing AIP Logic workflow created by a SL 4 developer available. Appropriate configuration access granted by Data Steward. Workflow purpose and expected behavior documented by the SL 4 developer.

STANDARDS: Builder correctly configures workflow parameters, connects data sources, tests with representative sample data, and documents the configuration for future reference.

EQUIPMENT: MSS platform access; AIP Logic configuration access; documented workflow specification from the SL 4 developer.

DURATION: 2–3 hours.

PROCEDURE:

Step 1 — Configure workflow parameters: 1. Navigate to AIP Logic in the platform. 2. Open the target workflow. Verify you have configuration access (not just view access). 3. Review the workflow's purpose and expected behavior as documented by the SL 4 developer. 4. Under **Configuration**, review each configurable parameter: - Data source connections (which Object Types or datasets the workflow reads) - Output target (where the workflow writes its results) - Prompt configuration (if the workflow uses configurable natural language prompts — these can be adjusted for operational context without writing code) - Scheduling: on-demand, triggered by data change, or scheduled interval - Confidence threshold: the minimum confidence score before an output is surfaced to users 5. Make parameter changes according to the documented operational requirements. 6. Save the configuration.

Step 2 — Update data source connections: 1. In the workflow configuration, navigate to **Data Sources**. 2. Review existing connections — verify each connection points to the correct Object Type or dataset. 3. To update a connection: select the connection, choose the replacement Object Type or dataset, verify the field mapping (workflow input fields mapped to data source properties). 4. If the replacement data source has a different schema than the previous source, the field mapping will need to be reconfigured. Do not deploy a workflow with unmapped required fields. 5. Save and test with a sample data query before activating in production.

Step 3 — Establish monitoring schedule:

Active AIP Logic workflows require monitoring. Advanced builders are responsible for first-line monitoring of workflows they have configured.

Task	Frequency	Action on Issue
Review workflow execution logs	Weekly	Identify errors or failed executions
Spot-check AI output quality	Bi-weekly	Compare AI outputs to known-correct answers on sample records
Verify data source freshness	Weekly	Confirm source data is updating on schedule
Review user feedback flags	As received	Investigate flagged outputs, report to SL 4 developer
Check confidence score distribution	Monthly	Significant shift may indicate data drift

6-3. If monitoring reveals a workflow producing systematically incorrect outputs, deactivate it immediately and escalate to the SL 4 developer and Data Steward. Do not leave an incorrect AI workflow active in production while investigating — it will continue to surface bad outputs to users.

TASK 6-2: CONFIGURE NATURAL LANGUAGE QUERY ON AN OBJECT TYPE

TASK: Configure a natural language query interface on a Workshop application, defining data scope, operational prompt guidance, output format, and source citation, and test with representative operational questions.

CONDITIONS: Builder has completed SL 1 and SL 2 and is SL 3 qualified. AIP Logic query interface created by a SL 4 developer, available for configuration. Data Steward has authorized the data scope for natural language query access.

STANDARDS: Builder configures scope correctly to least-privilege, adds prompt guidance appropriate to the operational context, enables source citation, and tests with at least five representative operational questions before activating in production.

EQUIPMENT: MSS platform access; AIP Logic configuration access; authorization from Data Steward for data scope.

DURATION: 2–3 hours including testing.

> **NOTE**
>
> Operators (TM-10, Task 4-8B, Interact with an AIP Agent) validate AI outputs against source data before acting. When configuring prompts, design with that human-review requirement in mind. Prompts should produce outputs that operators can quickly verify. Refer to TM-10, Task 4-8B for the validation workflow operators follow — then design your prompts to produce outputs compatible with that workflow.

PROCEDURE:

1. Navigate to the AIP Logic query interface configuration.
2. Under **Data Scope**, define which Object Types and datasets the interface can query. Apply least-privilege — only include data the intended users are authorized to access.
3. Under **Prompt Guidance**, configure operational context that helps the AI produce relevant responses. Examples: a glossary of command-specific terminology, unit abbreviations, or equipment codes.
4. Configure **Output Format**: what format should responses take? Tabular, narrative, a combination?
5. Enable **Source Citation**: configure the interface to cite the specific records it used to produce each answer, enabling user verification.
6. Test with representative operational questions. Verify responses are accurate, appropriately scoped, and clearly labeled as AI-generated.
7. Document the data scope and prompt configuration for future reference and Data Steward audit.

TASK 6-3: BUILD AND CONFIGURE A BASIC AIP AGENT

TASK: Navigate to AIP Agent Studio, build a basic agent from scratch against an existing Ontology, configure its data sources and output instructions, test it against operational queries, and document the configuration. Additionally, review and adjust parameters on an existing agent.

CONDITIONS: Builder has completed SL 1 and SL 2 and is SL 3 qualified. At least one well-structured Object Type with populated data is available in a development environment. Builder has Agent Studio author access in the development project. Data Steward has approved the data sources the agent will query.

STANDARDS: Builder creates a functioning agent connected to at least one Object Type, writes a clear output instruction block, tests the agent with at least three operational queries, evaluates output quality against source data, and produces a configuration document. Builder can also locate an existing agent, review its configuration, and make authorized parameter adjustments.

EQUIPMENT: MSS platform access; AIP Agent Studio author access (development environment); Object Types with populated data; Data Steward approval for data sources.

DURATION: 4–6 hours.

> **NOTE**
>
> SL 3 vs. SL 4H Scope — SL 3 covers: creating basic agents using Ontology Object Types as data sources, writing output instructions, and validating agent responses. SL 4H covers: advanced prompt engineering, custom tool development (TypeScript/Functions on Objects), agentic action logic (agents that write back to objects or trigger workflows), multi-agent orchestration, production deployment, and agent monitoring at scale. If your use case requires agents that take actions — not just answer questions — escalate to a SL 4H developer.

PROCEDURE:

Step 1 — Navigate to Agent Studio:

1. From the platform navigation, search for "Agent Studio" or locate it in the AIP section of the platform.
2. Open Agent Studio. A list of existing agents in your environment appears.
3. Before creating a new agent, review at least one existing agent to understand how they are structured (name, data sources, output instructions).

Step 2 — Create a new agent:

1. Select **New Agent**. Assign a descriptive name following the command naming convention (e.g., `[UNIT]_[PURPOSE]_Agent_DEV`).
2. Write a clear agent description: what question does this agent answer, for which user population, and using what data.
3. Set the environment to **Development**. Do not publish directly to production.

Step 3 — Connect data sources:

1. Navigate to the **Context Sources** section.
2. Add the approved Object Types the agent will reason over. Select only Object Types that have been authorized by the Data Steward.
3. For each Object Type added, verify:
4. Properties are clearly named (the agent uses property names in its reasoning)
5. Properties have descriptions where the name alone is ambiguous
6. Data in the Object Type is current and correctly populated

7. Add any approved documents or datasets as supplementary context if required by the use case.

> **NOTE**
>
> Ontology quality directly determines agent quality. An agent connected to an Object Type with poorly named properties or missing data will produce poor responses. Fix Ontology issues before deploying the agent.

Step 4 — Write output instructions:

1. Navigate to the **Output Instructions** (sometimes labeled System Prompt or Agent Instructions depending on platform version).
2. Write output instructions covering:
3. **Role:** What is this agent? (Example: "You are a data assistant for USAREUR-AF staff sections. Answer questions about [Object Type] data accurately and concisely.")
4. **Scope:** What should the agent answer, and what should it decline? (Example: "Answer only questions that can be addressed using the connected data sources. If a question cannot be answered from available data, say so clearly.")
5. **Format:** How should responses be structured? (Example: "Respond in plain English. Use bullet points for lists. Always cite the Object name or dataset when referencing specific data.")
6. **Limitations:** What must the agent always state? (Example: "Always note that outputs are AI-generated and require human review before operational use.")
7. Keep output instructions concise. Long, complex instruction blocks at SL 3 level are a signal the agent's scope is too broad — narrow the use case.

> **CAUTION**
>
> Do not instruct the agent to present AI-generated outputs as confirmed facts. All agents must communicate that responses require human verification. This is a non-negotiable output instruction requirement.

Step 5 — Test the agent:

1. Open the **Test** interface within Agent Studio.
2. Enter at least three representative operational queries — the type of question a user would realistically ask in production.
3. For each response, evaluate:
4. **Accuracy:** Does the response match the actual data in the connected Object Types? Verify by checking the source objects directly.
5. **Source citation:** Does the agent correctly identify where the information came from?

6. **Scope adherence:** Does the agent stay within the data it is authorized to use, or does it speculate beyond available data?
7. **Format compliance:** Is the response formatted as instructed?
8. If the agent produces incorrect, out-of-scope, or unsupported responses, adjust the output instructions or data sources and retest. Do not proceed to sharing or publication if the agent fails any accuracy check.

Step 6 — Adjust parameters on an existing agent (separate task):

SL 3 builders are authorized to adjust the following parameters on agents created by others:

Parameter	SL 3 Authority	Notes
Context sources	Yes	Add or remove approved Object Types or datasets
Output instructions	Partial	Adjust operational terminology, formatting, unit-specific language
Linked Object Types	Yes	Within authorized data scope and with Data Steward approval
Agent tools / action logic	No — SL 4H	Do not modify — escalate
System prompt reasoning logic	No — SL 4H	Do not modify — escalate

1. When adjusting an existing agent, document the pre-change configuration before editing.
2. Make one change at a time and retest after each change to isolate the effect.

Step 7 — Document and report:

1. Produce a configuration document for the agent covering: purpose, data sources, output instruction summary, test queries and results, and any known limitations.
2. Submit the configuration document to the Data Steward for review before sharing the agent with end users.
3. Do not publish the agent to production. Production deployment is SL 4H scope — coordinate with a SL 4H developer to transition a validated agent to production.

How builder-side Ontology quality affects agent performance:

AIP agents reason over the properties and values on the Object Types they are connected to. If Object Type properties have ambiguous names, missing descriptions, or incorrect data types, the agent will produce lower-quality responses. The single most impactful action a SL 3 builder can take to improve agent quality is ensuring well-named, well-described, correctly typed properties on all linked Object Types. Fix the Ontology first; tune the agent second.

6-4. AI FDE (AI-DRIVEN FEATURE DEVELOPMENT ENVIRONMENT)

AI FDE is Palantir's platform capability for code-assisted AI product development, integrating AI tooling directly into the Foundry development workflow. AI FDE supports developers in building AI-enabled Foundry applications by providing AI-assisted code generation, testing, and iteration within the platform's development environment.

SL 3 scope: awareness only. AI FDE represents the direction of the AIP engineering toolchain and is relevant context for advanced builders who work alongside SL 4 developers. Understanding what AI FDE is — and what it is for — helps SL 3 builders communicate effectively with SL 4 engineers and understand the development environment those engineers are working in.

Reference: *Product Launch: AI FDE | DevCon 3* — available on the Palantir Developers YouTube channel (@PalantirDevelopers) — provides an orientation to the AI FDE platform direction and the AIP engineering toolchain. SL 3 builders are encouraged to watch this reference for situational awareness.

Hands-on development with AI FDE is SL 4H (AI Engineer) territory. SL 3 builders do not configure, use, or manage AI FDE directly.

CHAPTER 7 — DATA GOVERNANCE AND LINEAGE

BLUF: Advanced builders are frontline data governance actors — they read and understand lineage graphs, identify and report data quality issues, work with Data Stewards on resolution, and enforce governance standards in everything they build.

NOTE

SL 2 builders follow governance standards defined in TM-20, Chapter 8 (Builder Standards and Governance). SL 3 builders have additional stewardship responsibilities because your designs affect shared infrastructure and downstream systems. At SL 3 level you are responsible for: (1) understanding operator access expectations (TM-10, Chapter 6, Security, Classification, and Markings); (2) ensuring SL 2 builders can implement your designs without overstepping their scope; (3) coordinating with data stewards before modifying any shared production resource; (4) designing for the full downstream impact across all consumers, not only the immediate use case.

NOTE

Audit Trail — All advanced builder actions in MSS — Ontology modifications, pipeline builds, branch creation, peer reviews, and production promotions — are logged with your credentials, timestamp, and the specific resource changed. These logs are used for accountability reviews, change tracking, and incident investigation. You are personally accountable for all changes made under your credentials, including changes made on behalf of another team member.

NOTE

CUI (Controlled Unclassified Information) — Before designing any data product that ingests or exposes CUI, coordinate with your data steward and confirm: (a) the data is authorized for ingestion, (b) access controls are configured to restrict visibility to authorized personnel, and (c) any downstream applications exposing CUI are reviewed before publication. Coalition-facing data products require additional coordination with C2DAO and, where applicable, NATO data governance review.

7-0a. Q1 2026 Lineage and Governance Feature Updates

The following capabilities were added to the platform lineage and governance toolset in Q1 2026:

Presentation mode for lineage graphs. Lineage graphs now support a presentation mode that renders a clean, full-screen view of the dependency graph — suitable for Data Steward briefings, peer reviews, and command presentations. Activate presentation mode from the **View** menu within any lineage graph.

Use this mode when briefing downstream impact assessments (Task 7-1) to commanders or governance boards. Presentation mode suppresses technical metadata and highlights the data flow path for non-technical audiences.

Multi-ontology support. The lineage view now traces dependencies across multiple ontologies within the same environment. For USAREUR-AF builders operating in environments where multiple ontologies coexist (e.g., theater-level ontology and coalition-shared ontology), lineage graphs display cross-ontology links and dependencies. This is critical for downstream impact assessments — a change in one ontology may affect Object Types, pipelines, or applications in another. When performing Task 7-1, verify that the lineage graph is displaying cross-ontology dependencies by confirming the **Multi-Ontology** toggle is enabled.

Log search functionality. A log search capability is now available within the lineage and governance interface. Builders can search execution logs, audit trails, and change histories by keyword, resource name, date range, or user. Use log search to: - Investigate when a specific resource was last modified and by whom - Trace the execution history of a pipeline or automation across a defined time window - Support data quality investigations (Task 7-2) by identifying when a data issue first appeared

Access log search from the **Logs** panel within lineage or from the platform's global search interface.

Monitoring status color modes. Lineage graphs now support configurable color modes for monitoring status — enabling builders to visualize resource health at a glance. Available modes include: health status (green/amber/red based on execution success rate), freshness (color-coded by time since last update), and access level (color-coded by permission tier). Select the color mode from the lineage graph toolbar. Use health status mode during routine monitoring and freshness mode when investigating stale data issues.

Expanded access across the platform. Lineage views are now accessible from additional platform entry points — including directly from Workshop applications, Contour analyses, and Object Explorer — without navigating to the dedicated lineage tool. This reduces friction for builders performing ad hoc impact assessments. When viewing any resource, look for the **Lineage** icon in the resource header to access the lineage graph in context.

TASK 7-1: PERFORM A DOWNSTREAM IMPACT ASSESSMENT USING LINEAGE

TASK: Use the platform lineage graph to identify all downstream dependencies of a resource targeted for modification, notify downstream owners, and document findings before executing any change.

CONDITIONS: Builder has completed SL 1 and SL 2 and is SL 3 qualified. Access to the Lineage view for a dataset, Object Type, or pipeline. Understanding of the upstream data sources that feed the product. Planned modification to a shared resource identified.

STANDARDS: Builder correctly interprets a lineage graph, identifies all upstream dependencies, lists all downstream consumers, notifies downstream owners, and documents findings clearly for Data Steward review.

EQUIPMENT: MSS platform access; Lineage view access on the target resource.

DURATION: 1–2 hours.

PROCEDURE:

Step 1 — Read the upstream lineage graph: 1. Navigate to the target dataset or Object Type. 2. Open the **Lineage** tab or view. 3. Identify the immediate upstream inputs (what datasets or transforms produced this one). 4. Follow each branch upstream to the source — find the original ingestion points. 5. Note the names and owners of all upstream datasets. 6. Identify any external sources — these are data outside the platform and may have different quality, freshness, and reliability characteristics. 7. Document findings: "This Object Type is produced by pipeline X, which joins dataset A (owned by S4) and dataset B (owned by G4 logistics). Dataset A originates from the PBUSE feed ingested daily at 0600Z."

Lineage graph elements reference:

Element	Shape/Color	Meaning
Source dataset	Rectangle, source icon	Raw or ingested data — where data enters the platform
Pipeline	Arrow with transform icon	A Pipeline Builder or code transform that produced an output
Output dataset	Rectangle	A derived dataset produced by a pipeline
Object Type	Object icon	An Ontology Object Type backed by a dataset
External source	Cloud icon	Data from outside the platform (file upload, external feed)
Edge (arrow)	Directional arrow	Data flows from source to target

Step 2 — Perform the downstream impact assessment: 1. Open the lineage graph for the resource you plan to modify. 2. Switch to **Downstream** view — this shows all resources that depend on this one. 3. List all downstream datasets, Object Types, Workshop applications, Contour views, and Quiver analyses. 4. For each downstream resource, identify the owner and notify them of the planned change. 5. Assess whether the change is breaking (schema change, property rename, type change) or non-breaking (new column added, description updated). 6. For breaking changes: coordinate a maintenance window, get Data Steward approval, and notify all downstream owners before executing. 7. For non-breaking changes: notify downstream owners as a courtesy and document the change.

WARNING

Renaming a property on an Object Type is a breaking change for every Workshop application, Contour view, Quiver analysis, and AIP Logic workflow that references that property by name. Never rename a production property without a full downstream assessment and coordinated cutover plan.

TASK 7-2: EXECUTE A DATA QUALITY INVESTIGATION

TASK: Identify, document, and report a data quality issue, trace it to its source using the lineage graph, and coordinate with the Data Steward for resolution.

CONDITIONS: Builder has completed SL 1 and SL 2 and is SL 3 qualified. Identified data quality issue in a dataset or Object Type. Data Steward contact information available.

STANDARDS: Builder correctly identifies and documents a data quality issue, reports it through the appropriate channel, traces probable source, and follows up until resolution is confirmed.

EQUIPMENT: MSS platform access; Lineage view access; data quality issue tracker or C2DAO issue log access.

DURATION: 1–3 hours.

> **NOTE**
>
> Operators (TM-10, Chapter 5, Working with Data) expect data to be current, accurate, and correctly marked. Use TM-10, Task 5-4 (Verify Data Currency and Source) as the benchmark for what operators require from your data products. If your pipeline or Ontology design creates data quality issues that operators cannot resolve themselves (TM-10, Chapter 5-2, What to Do When Data Looks Wrong), you are responsible for identifying and fixing the root cause.

PROCEDURE:

Step 1 — Identify and classify the issue:

Issue Type	Example	Severity
Missing data	Key field is NULL on 30% of records	High — analysis results are incomplete
Duplicate records	Same vehicle appears twice with different status	High — counts and aggregations are incorrect
Stale data	Fleet dataset last updated 5 days ago; should update daily	Medium — analysis is out of date
Invalid values	Status field contains "n/a" when valid values are FMC/PMC/NMC	Medium — filters and charts break
Schema mismatch	Column renamed in upstream source, pipeline now fails	High — output not updating

Issue Type	Example	Severity
Referential integrity	Vehicle records reference unit UICs that don't exist in the unit dataset	Medium — joins produce NULLs

Step 2 — Report the issue: 1. Document the issue with specificity: - What is the affected dataset or Object Type (full name)? - What is the issue? Be specific: "15% of records in the LogVehicle Object Type have NULL values in the `assigned_unit_uic` property as of 11 March 2026." - What is the operational impact? "This causes Quiver unit-to-vehicle link analysis to omit approximately 200 vehicles from unit rollups." - What is the likely source? Trace upstream using the lineage graph. - When was the issue first observed? 2. Identify the responsible Data Steward from the C2DAO registry. 3. Submit the issue report through the approved governance channel (data quality issue tracker, C2DAO issue log, or direct coordination with Data Steward per local SOP). 4. Retain a copy of the report with timestamp. 5. Do not attempt to fix a data quality issue in another team's dataset without explicit authorization from the Data Steward — you may be working with a copy or a view, not the authoritative source.

Step 3 — Routine Data Steward coordination: - **Before creating a new Object Type or dataset:** confirm naming convention, verify the entity is not already modeled elsewhere - **Before modifying a shared resource:** notify and get approval - **When reporting a quality issue:** submit through the Data Steward's designated channel - **Before publishing a production application:** request governance review - **When assigning access:** confirm the access level is appropriate and approved

If you cannot identify the Data Steward for a dataset or Object Type, escalate to the C2DAO authority for your domain. Do not proceed with modifications to unowned data without coordination.

TASK 7-3: CONFIGURE ACCESS CONTROLS ON A WORKSHOP APPLICATION

TASK: Configure sharing and permissions on a Workshop application, verify least-privilege access for the intended audience, and document the access configuration.

CONDITIONS: Builder has completed SL 1 and SL 2 and is SL 3 qualified. Workshop application, dataset, or Object Type requiring access configuration. Appropriate administrative rights confirmed. Access requirements documented and approved by Data Steward.

STANDARDS: Builder configures access using least-privilege principle, documents the access rationale, and verifies the configuration grants access to intended users and denies access to all others before publishing.

EQUIPMENT: MSS platform access; Workshop admin rights on the target application; Data Steward authorization.

DURATION: 30–60 minutes.

> **NOTE**
>
> Before designing coalition-facing data products at SL 3 level, understand TM-10, Chapter 6 (Security, Classification, and Markings), especially Task 6-1 (Verify Markings and Access Level). Coalition data must be correctly marked and access-controlled from ingestion through final product. Errors in releasability markings can result in data shared with unauthorized coalition partners — this is a hard governance gate, not a best practice. Coordinate with the USAREUR-AF C2DAO before any coalition-facing design decision.

PROCEDURE:

Step 1 — Answer the three access control questions for every resource: 1. **Who needs to see this?** Configure view access for those users or groups only. 2. **Who needs to edit this?** Configure edit access for those users or groups only — typically just you and your Data Steward at initial publication. 3. **Who should never see this?** Verify that access is not inherited from a broader group that includes unauthorized personnel.

Access control levels on the platform: - **Dataset level:** who can read or write the underlying dataset - **Object Type level:** who can view objects, who can execute Actions - **Application level:** who can view the Workshop application, who can edit it - **Analysis level:** who can view a Contour or Quiver saved view

Step 2 — Set access controls on the Workshop application: 1. Open the application in edit mode. 2. Navigate to **Sharing and Permissions** (application settings). 3. Remove any default "all users" access if the application contains controlled data. 4. Add specific users or groups with the appropriate role: - **Viewer:** can interact with the published application, cannot edit - **Editor:** can modify the application in edit mode - **Owner:** full administrative control 5. If the application references Object Types or datasets with restricted access, verify that the access controls on those underlying resources match the application's access. Granting application view access does not grant underlying data access — both must be configured. 6. Document the access configuration in the application description: "Authorized for USAREUR-AF G4 staff (viewer), G4 data team (editor)." 7. Test access by verifying a member of each access group can interact as expected.

> **NOTE**
>
> Additional access management reference (self-study) — Two Palantir Developers reference videos extend the access control content in this chapter: *Security | How to use Projects to Help Enable your Business to Scale* (using Foundry Projects as the primary access boundary for scaling data products to many users) and *Security | How to Debug a User's Access to a File* (diagnosing why a specific user cannot access a resource — useful when access complaints arrive from operators). Both are UI-based procedures within SL 3 scope.

CHAPTER 8 — ENVIRONMENT MANAGEMENT

BLUF: Advanced builders manage the full development lifecycle — creating branches for development work, reviewing changes, and executing the promotion workflow to production — entirely through the platform UI without scripting or CI/CD configuration.

8-1. BRANCHING AND THE DEVELOPMENT LIFECYCLE

NOTE

SL 2 builders follow a development lifecycle defined in TM-20, Chapter 7 (Branching and Environment Management): develop on a branch, test, request merge, get approval, merge to main. SL 3 development follows the same pattern with more rigorous testing gates because your changes affect shared infrastructure. Operators (SL 1) only access the main/production branch. Every merge to main is a production release. Apply engineering discipline — test against SL 1 operator workflows (Chapter 4) before merging.

8-1. The platform uses a branching model to separate development work from production. A branch is an isolated copy of the environment where changes can be made, tested, and reviewed without affecting the production environment that other users depend on.

8-2. The three environments for USAREUR-AF builders:

Environment	Purpose	Who Works Here
Production (main branch)	The live environment used by all consumers	No one develops here; promotion only
Development branch	Where new features and changes are built	The builder working on the change
Review branch (pre-production)	Staging area for Data Steward and peer review before promotion	Reviewer and Data Steward

8-3. Never develop directly on the production branch. If you are making changes in production, you are doing it wrong.

8-1a. Q1 2026 Branch Security Updates

Role-based security for branches. The platform now supports role-based access control on individual branches. Branch creators and administrators can assign users specific roles — such as Viewer, Developer, or Manager — to control who can view, commit to, review, and merge a given branch. This capability replaces the previous model where branch access was controlled solely through project-level permissions.

When creating a branch (Task 8-1), configure branch roles under **Branch Settings > Security**:

Branch Role	Permissions
Viewer	Can view branch contents; cannot commit changes
Developer	Can view and commit changes; cannot merge to production
Manager	Full control — can commit, review, approve, and merge

Assign the minimum role necessary for each user. Builders working on their own development branch should be assigned the Developer role; the reviewing Data Steward or peer reviewer should be assigned the Manager role to authorize promotion.

Upgraded branch security enabled for all users. As of Q1 2026, upgraded branch security is enabled platform-wide for all users. This means role-based branch permissions are no longer an opt-in feature — they are the default. All existing branches have been migrated to the new security model. Builders who previously relied on project-level permissions alone must verify that their branch-level roles are correctly configured. Review branch security settings on all active development branches and confirm that roles align with your team's access requirements. Report any access discrepancies to the Data Steward.

NOTE

Branch security roles are separate from application-level and Object Type-level access controls (Task 7-3). A user with Manager role on a branch can merge changes to production, but the promoted resources still respect their own access controls. Branch security governs who can develop and promote; resource-level security governs who can consume.

TASK 8-1: CREATE AND CONFIGURE A DEVELOPMENT BRANCH

TASK: Create a named development branch from the production branch, configure workspace to the branch, and verify isolation from production before beginning any development work.

CONDITIONS: Builder has completed SL 1 and SL 2 and is SL 3 qualified. Identified change to implement. Data Steward awareness of the planned change confirmed.

STANDARDS: Builder creates a named branch with a descriptive name, completes all development in the branch, and does not merge to production without peer review and Data Steward sign-off.

EQUIPMENT: MSS platform access; Branch Manager access.

DURATION: 30 minutes.

PROCEDURE:

1. Navigate to the **Branch Manager** or **Environment Manager** in the platform.
2. Select **Create Branch** from the production branch (or from main).
3. Name the branch using the convention: `[builder-id]_[change-description]_[YYYYMMDD]`
   Example: `sgt-jones_readiness-dashboard-v2_20260311`
4. Confirm the branch is created from the correct base (production).
5. Switch to the new branch in your workspace — verify you are working in the branch, not in production.
6. While working in the branch:
7. Make all changes (pipeline updates, Workshop edits, ontology changes) in the branch only
8. Test all changes thoroughly in the branch
9. Do not promote until all tests pass and the change is ready for review

---

TASK 8-2: CONDUCT A PEER REVIEW

TASK: Submit a completed development branch for peer review, or conduct a review of a peer's branch, ensuring all changes are tested and documented before authorizing promotion.

CONDITIONS: Builder has completed SL 1 and SL 2 and is SL 3 qualified. Completed development in a branch. All changes tested in the development branch. Ready for review.

STANDARDS: Builder submits a review request with complete documentation of changes. Reviewer examines all changes, tests functionality, and provides written approval or documented change requests before promotion is authorized.

EQUIPMENT: MSS platform access; Branch Manager access; review request system.

DURATION: 1–3 hours depending on scope of changes.

PROCEDURE:

As the builder submitting for review: 1. In the Branch Manager, locate your development branch. 2. Select **Request Review** or **Create Review Request**. 3. Complete the review request form: - **Summary of changes:** what was built or modified? - **Test results:** what did you test? What was the outcome? -

Downstream impact: what downstream products are affected? - **Rollback plan:** if the change breaks something in production, how will it be reversed? - **Data Steward notification:** confirm the Data Steward has been notified. 4. Assign the reviewer (a qualified peer or the Data Steward). 5. Submit the request.

As the reviewer: 1. Open the review request. 2. Switch to the branch being reviewed. 3. Examine all changes: - Pipeline changes: verify logic, verify output schema, run a test execution and check results. - Workshop changes: test all interactive elements, verify variable behavior, check all pages. - Ontology changes: verify naming conventions, check downstream impact. - Access controls: verify access is configured correctly. 4. Document your review findings. 5. Approve the review (no issues found) or request changes (document specific items to address before promotion).

TASK 8-3: PROMOTE CHANGES TO PRODUCTION

TASK: Execute the promotion of a reviewed and approved development branch to the production environment, verify production health after promotion, and notify downstream owners.

CONDITIONS: Builder has completed SL 1 and SL 2 and is SL 3 qualified. Approved review documented. Data Steward sign-off obtained. Scheduled promotion window coordinated with downstream owners.

STANDARDS: Builder executes promotion during the authorized window, confirms production environment is healthy after promotion, and notifies downstream application owners within one hour of completion.

EQUIPMENT: MSS platform access; Branch Manager access; all review approvals on file.

DURATION: 30–60 minutes plus post-promotion verification.

PROCEDURE:

1. Confirm all review approvals are recorded.
2. Confirm Data Steward has provided sign-off.
3. Coordinate with downstream application owners — if the promotion will cause a brief downtime or change a data schema, notify consumers in advance.
4. Schedule the promotion during an approved maintenance window if the change is significant.
5. In the Branch Manager, select the reviewed branch and initiate **Promote to Production**.
6. Monitor the promotion progress in the platform logs.
7. After promotion completes:
8. Verify the production environment is healthy: open the promoted application, run a pipeline, check that the promoted changes are visible and correct.
9. Verify that other applications that share the changed resources are still functioning.
10. Notify downstream owners that the promotion is complete.

11. If issues are identified post-promotion, execute the rollback plan immediately. Do not attempt to fix issues with additional changes while production is impaired — rollback first.

WARNING

Promoting a change that breaks a downstream application or pipeline in production is a significant governance event. Report the incident to the Data Steward immediately. Follow the post-incident review process to document root cause and corrective action.

8-2. PRODUCTION DISCIPLINE

8-4. Production discipline is the set of behaviors that maintain trust in the production environment. Advanced builders are directly responsible for production discipline in everything they promote.

8-5. Production discipline standards:

Standard	Requirement
No direct production edits	All changes through branches with review — no exceptions
Test before review	All changes tested in the development branch before requesting review
Review before promotion	No self-approved promotions — another qualified individual must review
Document all promotions	Maintain a log of what was promoted, when, by whom, and for what purpose
Notify downstream owners	Any promotion that affects shared resources requires advance notification
Rollback plan	Every significant promotion must have a documented rollback procedure
Incident reporting	Any production issue caused by a promotion must be reported within 24 hours

CHAPTER 9 — AUTOMATIONS *(SELF-STUDY / REFERENCE)*

NOTE

This chapter is self-study material. It is not included in the 5-day SL 3 course schedule. Instructors may demo Automations during Day 4 if time permits. Builders should complete this chapter independently before configuring Automations in a production environment.

BLUF: Automations are Foundry's capability to trigger actions, pipeline runs, or object updates on a schedule or based on object state changes — configured through Ontology Manager. They eliminate manual, recurring operational data tasks and enable the platform to maintain data currency and state without human intervention for routine processes.

9-1. WHAT AUTOMATIONS ARE

An Automation is a configured rule in Ontology Manager that monitors an Object Type's state or a schedule, and executes a defined action when the trigger condition is met. Automations operate in the background — builders configure them, and the platform executes them without operator involvement.

When to use Automations:

Use Case	Example
Recurring state updates	Automatically flag equipment as overdue for inspection when the last inspection date exceeds 30 days
Scheduled property resets	Reset a "submitted today" flag on all SITREP objects at 2359 daily
Alert conditions	Write a flag property to "ALERT" when a readiness threshold drops below a configured value
Pipeline trigger	Trigger a curated dataset rebuild when an upstream source object changes
Notification	Set a notification flag property when an approval action has been pending for more than 24 hours

When NOT to use Automations:

Automations execute at the Object Type level — they apply a rule to objects meeting a condition. They are not designed for complex multi-step process flows, handoff routing, or end-to-end workflow management. For processes involving multiple roles, conditional routing, and process state visibility, see Chapter 10 (Machinery).

9-1a. Q1 2026 Automations Feature Updates

The following capabilities were added to Foundry Automations in Q1 2026 and are within SL 3 configuration scope:

Interface parameters in Automate actions. Automate actions now support interface parameters — enabling builders to pass structured input values when an automation triggers an Action on an Object Type. Previously, automated actions could only execute with static or object-derived values. With interface parameter support, builders can configure automations that pass dynamic input values defined by an interface (see Task 4-4, Define an Interface on an Object Type). When configuring an Automate action (Task 9-1, Step 3), select **Action with Interface Parameters** and map the interface fields to the appropriate source values (object properties, static values, or computed expressions). This enables more flexible automation logic without requiring SL 4 code-level development.

Function effects in Automate. Automations can now trigger function effects — platform-defined side effects that execute as part of an automation action. Function effects extend the range of actions available to automations beyond simple property updates and object creation. Examples include triggering a notification, writing an audit log entry, or invoking a platform-registered function. At SL 3 level, configure function effects only when they have been pre-built and approved by a SL 4 developer. Do not author new function effects — that is SL 4 scope. When a function effect is available for use, it appears in the **Action** dropdown during automation configuration.

Streaming time series conditions. Automations now support trigger conditions based on streaming time series data. Builders can configure automations that fire when a time series property crosses a threshold, deviates from a trend, or meets a rate-of-change condition. This is operationally relevant for monitoring sensor feeds, readiness indicators, or logistics metrics that update continuously. When defining a trigger (Task 9-1, Step 2), select **Time Series Condition** and configure the property, evaluation window, and threshold. Coordinate with the Data Steward to confirm that the time series data source updates at sufficient frequency to support the intended trigger cadence.

Autopilot — visualization, monitoring, and debugging for automation workflows. Autopilot is a new platform tool for visualizing, monitoring, and debugging automation workflows. It provides a graphical view of all active automations on an Object Type — showing trigger conditions, action chains, execution status, and error states in a single dashboard. Use Autopilot to: - Visualize the full set of automations configured on an Object Type and their relationships - Monitor execution status in real time — identify failed, delayed, or stuck automations - Debug automation issues by inspecting trigger evaluation results and action execution logs - Identify conflicts between automations (e.g., two automations writing to the same property with different values)

Access Autopilot from the **Automations** panel on any Object Type or from the platform's global tools menu. Autopilot is a monitoring and diagnostic tool — it does not replace the configuration workflow in Task 9-1. Use Autopilot for ongoing monitoring (Task 9-1, Step 5) and for troubleshooting automation failures before escalating to a SL 4 developer.

TASK 9-1: CONFIGURE AN AUTOMATION ON AN OBJECT TYPE

TASK: Configure an Automation on an existing Object Type to trigger a defined action based on a schedule or object condition, test the automation in a development environment, and monitor its execution history.

CONDITIONS: Builder has completed SL 1 and SL 2 and is SL 3 qualified. Target Object Type exists and is correctly configured. Data Steward has authorized the automated write-back or property modification. Builder is working on a development branch.

STANDARDS: Builder correctly configures the automation trigger (schedule or condition), defines the action to execute, tests the automation in a development environment with representative data, verifies correct execution, and documents the automation in the Object Type's description field. Automation is approved by the Data Steward before promotion to production.

EQUIPMENT: MSS platform access; Ontology Manager edit permissions; Data Steward authorization for write-back; development branch active.

DURATION: 2–4 hours including testing.

> **WARNING**
>
> Automations execute automatically against live objects. An incorrectly configured Automation can modify property values on large numbers of objects simultaneously before a builder can intervene. Always test on a development branch with representative sample data. Never configure and immediately activate an Automation in production without testing.

PROCEDURE:

Step 1 — Open the Automations panel:

1. In Ontology Manager, navigate to the target Object Type (on your development branch).
2. Click the **Automations** tab (the location varies by platform version — it may also appear under **Rules**, **Triggers**, or an equivalent section).
3. Click **New Automation** or **Add Rule**.

Step 2 — Define the trigger:

1. Select the trigger type:

Schedule trigger: The automation runs at a defined interval regardless of object state. - Set the frequency: hourly, daily, weekly, or custom cron expression. - Set the time (UTC). Account for USAREUR-AF local time offset (UTC+1 in winter, UTC+2 in summer). - Specify whether the automation applies to all objects of this type or only objects matching a filter condition.

Object condition trigger: The automation runs when an object's property value meets a defined condition. - Select the property to monitor (e.g., `last_inspection_date`, `status`, `submission_flag`). - Define the condition operator (equals, greater than, less than, is null, changes to value). - Enter the threshold value (e.g., `last_inspection_date` is more than 30 days ago; `status` changes to "OVERDUE").

1. If using a condition trigger, define the evaluation frequency — how often the platform checks whether objects meet the trigger condition (minimum depends on platform configuration; confirm with Data Steward).

Step 3 — Configure the action to execute:

1. Under **Action**, select what happens when the trigger fires:
2. **Update Property:** Write a defined value to a specific property on the object. Example: set `inspection_status` to "OVERDUE".
3. **Create Object:** Create a new object of a specified type. Use for generating event or log records.
4. **Execute an existing Action:** Invoke an Action already configured on the Object Type (see TM-20, Task 4-6 and TM-30, Task 4-3).
5. **Trigger Pipeline Build:** Initiate a pipeline refresh. This option may require SL 4 coordination to configure correctly.
6. Configure the action parameters:
7. For **Update Property**: select the target property and specify the value to write (static value, or derived from another property on the same object).
8. For **Create Object**: specify the Object Type to create and map required properties from the triggering object.
9. Set the **Action Authorization**: the automation must execute under an authorized service account or role. Coordinate with the Data Steward to confirm the service account has write authorization for the affected Object Type.

Step 4 — Test the automation in a development environment:

1. Save the automation configuration on your development branch.
2. Identify a small set of test objects that should trigger the automation (objects that meet the trigger condition or that will be reached by the schedule).
3. Use the **Test** or **Dry Run** option (if available in your platform version) to simulate execution without writing changes. Verify the correct objects are selected and the correct values would be written.

4. If Dry Run is not available, manually trigger the automation on the development branch and verify results in Object Explorer.
5. Confirm that objects not meeting the trigger condition are not modified.
6. Confirm that the action writes the correct value to the correct property.

Step 5 — Monitor automation execution history:

1. After promotion to production (via the standard review and promotion workflow — Chapter 8, Task 8-3), navigate to the Automations panel on the Object Type.
2. Under **Execution History** or **Run History**, review past executions:
   - Status (success, failure, partial)
   - Number of objects affected per run
   - Timestamp and duration
3. Review the execution history at least weekly for the first two weeks after activating a new automation.
4. Set up failure alerting: configure your MSS notification settings or coordinate with the Data Steward to ensure pipeline or automation failures generate a notification.

Step 6 — Document the automation:

1. In the Object Type description field, add an entry documenting the automation:
   - What does it do?
   - What is the trigger?
   - What property does it modify?
   - When was it activated and by whom?
   - Who is the Data Steward for this automation?

NOTE

Automations that modify properties used by Workshop applications will affect what operators see in those applications. Before activating an automation on a production Object Type, notify the owners of all downstream Workshop applications that the property values will be changing automatically.

CHAPTER 10 — MACHINERY *(SELF-STUDY / REFERENCE)*

NOTE

This chapter is self-study material. It is not included in the 5-day SL 3 course schedule. Instructors may demo Machinery during Day 4 if time permits. Builders should complete this chapter independently before modeling processes in a production environment.

BLUF: Machinery is Palantir Foundry's business process management layer. It enables builders to model, monitor, and automate multi-step staff processes — approval workflows, recurring reporting cycles, data quality reviews — as structured, visible process flows connected to Ontology Object Types.

10-1. WHAT MACHINERY IS

Machinery manages end-to-end process flows across multiple steps and roles. Where Automations handle individual object-level triggers (one object meets a condition, one action executes), Machinery manages processes that span multiple objects, multiple people, multiple sequential steps, and require visibility into where in the process a given work item currently is.

The core difference:

Capability	Automations	Machinery
Scope	Single object, single trigger, single action	Multi-step process across objects and roles
Visibility	Execution history log	Live process state — "where is this item now?"
Routing	Not conditional	Conditional routing based on decisions, approvals, timeouts
Use case	Recurring property updates, state flags	Approval chains, sequential workflows, staffing processes

When to use Machinery:

- A process involves more than one step (e.g., data submission → review → approval → publication)
- A process involves handoffs between different roles or staff sections
- Command or operational leadership needs to see where items are in the process at any given time (process state visibility)

- A process has conditional outcomes (approved vs. rejected, or different routing based on data values)
- Recurring process cycles need to be triggered on schedule and tracked to completion

Examples of USAREUR-AF staff processes suited to Machinery:

- Weekly SITREP submission → G3 review → C2DAO validation → publication
- New equipment record creation → S4 review → G4 approval → Ontology publication
- Data quality issue report → assigned to data steward → investigated → resolved → closed
- Request for MSS access → unit data steward review → C2DAO approval → account provisioned

---

TASK 10-1: ORIENT TO MACHINERY AND MODEL A BASIC PROCESS

TASK: Navigate to Machinery, create a basic process definition with defined steps, roles, and transitions, link the process to existing Object Types, and monitor active process instances.

CONDITIONS: Builder has completed SL 1 and SL 2 and is SL 3 qualified. An operational process has been identified and documented by the Data Steward and process owner. Ontology Object Types representing the work items in the process exist and are correctly configured. Data Steward has authorized Machinery configuration for this process.

STANDARDS: Builder creates a process definition with at least two steps, at least two role assignments, and at least one conditional transition. Process is connected to the correct Object Type. A test process instance is initiated and successfully transitions through at least one step before branch submission for review.

EQUIPMENT: MSS platform access; Machinery access (confirm with Data Steward that Machinery is available in your command's MSS environment); Ontology Manager access; development branch active.

DURATION: 3–5 hours including testing.

PROCEDURE:

Step 1 — Navigate to Machinery:

1. Use the platform Search bar to find "Machinery." Alternatively, navigate to Machinery from the platform's application or product navigation.
2. Open Machinery. If you see a list of existing processes, review them to understand patterns before creating a new one.
3. Confirm you are on your development branch before creating any process definitions.

Step 2 — Create a basic process definition:

1. Click **New Process** or **Create Process**.

2. Name the process using a plain-English description of the staff process it models. Use a naming convention that identifies the domain and process: `[Domain] — [Process Name]` (e.g., `LOG — Equipment Record Approval`, `OPS — SITREP Submission and Review`).
3. Add a description explaining what operational process this models, who initiates it, who approves, and what the final outcome is.

Step 3 — Define process steps:

1. Add **Steps** (also called stages or phases depending on platform version) for each phase of the process:
2. Click **Add Step**.
3. Name the step using an imperative verb phrase: "Submit," "Review," "Approve," "Publish," "Reject."
4. For each step, define the **Assignee Role**: who is responsible for completing this step? Roles should correspond to MSS Groups (e.g., "Unit Data Stewards," "G4 Staff," "C2DAO Reviewers").
5. Define what the assignee must do to complete the step (check a box, fill in a review field, click Approve or Reject).
6. Repeat for each step in the process.

Step 4 — Configure transitions between steps:

1. Transitions define how a process moves from one step to the next.
2. Add a **Transition** from Step 1 to Step 2. Set the transition condition: what must be true for the process to move forward?
3. For approval workflows, configure **Branch Transitions**: if the reviewer clicks Approve, the process moves to "Approved" state; if they click Reject, the process moves back to "Resubmit" state.
4. Configure **Timeout Transitions**: if a step is not completed within a defined time window, the process can auto-escalate or notify a supervisor.
5. Test your transition logic on paper first: trace the process from initiation through every possible outcome (approved, rejected, timed out, escalated). Confirm the step and transition configuration produces all expected outcomes.

Step 5 — Link the process to Object Types:

1. Under **Object Type Binding** or **Data Connection**, select the Object Type whose objects will flow through this process. Each instance of a work item (e.g., each equipment record submitted for approval) corresponds to one object of this type running through the process.
2. Configure **Step-to-Property Mapping**: which property on the Object Type reflects the current process step? This allows Workshop applications and Quiver dashboards to show operators where each object currently is in the process (e.g., `approval_status` property shows "Pending Review," "Approved," "Rejected").
3. Save the process definition.

Step 6 — Monitor active process instances:

1. After promotion to production, return to Machinery to monitor active process instances.
2. The Machinery dashboard shows:
    - How many instances are currently at each step
    - Which instances are overdue (approaching or past their step timeout)
    - Completed instances and their outcomes (approved, rejected)
3. Review the Machinery dashboard at minimum weekly for critical operational processes, or daily for processes with short turnaround requirements.
4. For overdue instances, identify the assignee and notify them (via your unit's standard communication channels — Machinery may also support in-platform notification depending on configuration).

NOTE

Machinery requires that the Object Types and properties used in the process are correctly configured and stable before process design. Changing a property name or type on an Object Type that Machinery is bound to will likely break the process integration. Apply the downstream impact assessment process (Chapter 7, Task 7-1) before any schema change on an Object Type with a Machinery binding.

CHAPTER 11 — STANDARDS, CONVENTIONS, AND BEST PRACTICES

BLUF: Consistent standards across all products built by USAREUR-AF advanced builders create a coherent, maintainable data environment. This chapter specifies the naming conventions, design patterns, and quality standards that apply to everything built at SL 3 level.

11-1. NAMING CONVENTIONS

11-1. All resources created on the MSS platform by USAREUR-AF builders must follow C2DAO naming conventions. Non-compliant naming is a governance deficiency and will be flagged in Data Steward reviews.

Dataset naming: `[domain]_[source-or-description]_[output-type]_v[version]` Examples: - `log_pbuse-vehicle-fleet_cleaned_v1` - `ops_sitrep-events_aggregated-daily_v2` - `med_patient-encounters_anonymized_v1`

Pipeline naming: `[domain]_[input-description]_[output-description]_v[version]` Examples: - `log_vehicle-fleet-maintenance_readiness-join_v1` - `ops_sitrep-raw_dedup-aggregated_v3`

Object Type naming: PascalCase, singular noun, prefixed with domain abbreviation: `[DomainAbbrev][EntityName]` Examples: - `LogVehicle` — Logistics domain, Vehicle entity - `OpsUnit` — Operations domain, Unit entity - `MedFacility` — Medical domain, Facility entity - `G2IntelligenceReport` — G2 domain, Intelligence Report entity

Workshop application naming: `[Domain] - [Audience] - [Application Purpose]` Examples: - `LOG - G4 Staff - Equipment Readiness Dashboard` - `OPS - BCT CDR - Operations Summary Board` - `G2 - All Staff - Intelligence Overview`

Contour and Quiver saved views: `[domain]_[analysis-purpose]_[frequency-or-audience]` Examples: - `log_readiness-by-unit_weekly` - `g2_threat-trend_cdre-brief`

Pipeline Builder node labels (within the pipeline graph): Label every node. Format: `[action]_[subject]` using snake_case. Examples: `src_vehicle_fleet`, `filter_active_only`, `join_maintenance_records`, `agg_by_unit_category`, `out_readiness_rollup`

Variable naming (Workshop): `[domain]_[descriptor]_[type]` Examples: `ops_selected_unit_string`, `log_date_filter_date`, `g2_show_classified_boolean`

Domain Abbreviations:

Domain	Abbreviation	Responsible Staff
Logistics	log	G4 / S4
Operations	ops	G3 / S3
Medical	med	G4 / Surgeon
Intelligence	g2	G2 / S2
Signal/Comms	sig	G6 / S6
Civil Affairs	g9	Civil Affairs
Personnel	g1	G1 / S1
Finance	fin	G8

11-2. DESIGN BEST PRACTICES

Application design:

1. **Start with the question, not the data.** What decision does this application support? Design the layout around the answer, then connect the data.
2. **One page, one question.** Each page should answer one operational question clearly. Resist the urge to put everything on one page.
3. **Design for the user's time.** Operational users are often time-constrained. The most important information should be visible without any clicks or scrolling.
4. **Use progressive disclosure.** Show summary first, detail on selection. Do not display maximum detail by default.
5. **Empty states are content.** Design what the user sees when no data matches the filters. "No records match the selected filters" is better than a blank page.
6. **Label everything.** Every widget should have a clear title. Every metric tile should have a unit of measure. Every table column should have a readable header.
7. **Consistent color use.** Use platform-standard colors for status (red/amber/green). Do not invent a color scheme that conflicts with other applications in the environment.

Pipeline design:

1. **Profile before you build.** Before joining or aggregating, examine source datasets. Know the row counts, key distributions, and NULL rates before designing your pipeline.
2. **Filter early.** Apply date and key filters as early in the pipeline as possible to minimize the data volume processed through downstream transforms.

3. **Document join decisions.** For every join node, add a description explaining why this join type was chosen, what the join key is, and what the expected row count ratio is.
4. **Test with known data.** Validate pipeline output against records you know the expected result for. If you are building a readiness rollup, manually compute the expected output for one battalion and compare to pipeline output.
5. **Plan for NULL.** Every production dataset contains NULLs. Add explicit NULL handling after every source node.
6. **Readable graphs.** Arrange pipeline nodes so the flow direction is clear (left to right or top to bottom). Label all nodes. Group related transforms visually.

Ontology design:

1. **Model the entity, not the dataset.** An Object Type represents a real-world operational entity — not a row in a spreadsheet. Design properties around what the entity IS, not what columns happen to exist in the source data.
2. **Fewer, better properties.** Include only properties that downstream consumers will actually use. Unused properties add schema complexity and maintenance burden.
3. **Status fields need valid value documentation.** Any property with a defined set of valid values must document those values in the property description.
4. **Link Types represent real relationships.** Only create a Link Type if the relationship between two Object Types is operationally meaningful and will be traversed in downstream analysis or applications.
5. **Primary key is sacred.** The primary key must be truly unique and stable. Changing the primary key on a production Object Type is a major breaking change.

NOTE — Primary Key Design Rules (Palantir Best Practice) 1. The `id` (primary key) column **must be type string** — no exceptions. 2. PKs must be inherently unique, derived from the object's own properties only. 3. All object types require a dedicated `id` column separate from business identifiers. 4. Never infer object properties from identifier values. 5. Composite keys: keep readable (concatenated), never hashed.

Source: Palantir Developer Community — Ontology and Pipeline Design Principles

NOTE — Foundry Project Architecture Taxonomy (Palantir Best Practice)

Organize Foundry projects into five standard types:

Type	Naming Convention	Purpose	Access
Datasource	`Datasource - {Name}`	Raw ingestion, quality monitoring, clean datasets	Editors: data engineers only
Data Integration	`Integration - {Name}`	Schema definition, aggregation, health checks	Restricted

Type	Naming Convention	Purpose	Access
Ontology	`Ontology - {Name}`	Object Types, relationships, views	Editors: engineers/managers; Viewers: end users
Application	`Application - {Name}`	Workflows, user-facing tools	Collaborative
Sandbox	`[sandbox] Name`	Experimentation, training — no business data	Universal creator access

RBAC per project: OWNER (admin), EDITOR (build/modify), VIEWER (consume), DISCOVERER (names/lineage only).

Source: Palantir Developer Community — Ontology and Pipeline Design Principles

NOTE — Object Type Naming and Maturity Standards (Palantir Best Practice) - Use natural business language — no abbreviations, no technical jargon in type names. - **No versioned names** — `Message_v2` is prohibited. Create a new type or modify in place. - **No `[tag]` prefixes** — use Ontology object type groups instead. - Every object type must carry a **maturity status**: `Experimental`, `Active`, or `Deprecated`. - Assign a dedicated **point of contact** for each object type. - Timestamp properties: `{action}_at_timestamp`. Author properties: `{action}_by_user`. - FK naming: `{foreign_object_type}_id` or `{link_api_name}_{foreign_object_type}_id`.

Source: Palantir Developer Community — Ontology and Pipeline Design Principles

11-3. PERFORMANCE CONSIDERATIONS

11-2. Advanced builders are responsible for the performance characteristics of what they build. An application that loads slowly or a pipeline that takes hours degrades the operational environment for all users.

11-3. Performance checklist before publishing to production:

Check	How to Verify	Action if Failing
Application page load time	Preview in an incognito browser session; time the load	Reduce widget count, add filters with default values, paginate large tables
Object Set size	Check row count on Object Set widget	Add a default filter to limit initial load; implement pagination
Pipeline run time	Check pipeline execution metrics	Add early filters, verify partition pruning is active on date filters
Map widget object count	Preview map with production data	Limit objects displayed; add a filter requiring user selection before map populates

Check	How to Verify	Action if Failing
Table row count	Preview table with production data	Implement server-side pagination; add a mandatory filter before results appear

11-4. USAREUR-AF C2DAO GOVERNANCE CHECKLIST SUMMARY

11-4. Every product published to production must satisfy the following governance requirements:

Naming Compliance: - ☐ Dataset, pipeline, Object Type, and application names follow C2DAO naming conventions - ☐ All pipeline nodes are labeled descriptively - ☐ All Workshop variables follow the naming convention

Documentation: - ☐ Dataset and pipeline descriptions are complete (source, purpose, owner, refresh cadence) - ☐ Object Type description explains the entity, source, and operational purpose - ☐ Object Type properties have descriptions for non-obvious fields - ☐ Actions have clear user-facing labels and confirmation messages

Access Control: - ☐ Access is configured to least-privilege for the intended audience - ☐ No unintended "all users" access on products containing sensitive or controlled data - ☐ Access configuration is documented in the product description

Quality: - ☐ NULL handling is implemented in all pipelines - ☐ Join types are documented and correct - ☐ Action validation rules are configured for all user-input parameters - ☐ Downstream impact assessment completed for any change to shared resources

Review and Promotion: - ☐ Development completed in a branch (not production) - ☐ Peer review completed and documented - ☐ Data Steward review and sign-off obtained - ☐ Promotion window coordinated with downstream owners

APPENDIX A — ADVANCED BUILDER CHECKLISTS

NOTE

Before initiating a SL 4 handoff, confirm the requirement genuinely exceeds SL 3 capability. Use this checklist: (1) Can this be built using SL 2 no-code tools? If yes — hand back to SL 2 builder, do not escalate to SL 4. (2) Can this be designed using SL 3 UI tools (Workshop, Ontology Manager UI, Pipeline Builder UI, AIP Logic UI)? If yes — build at SL 3 level, do not escalate. (3) Does implementation require writing code (Python, PySpark, TypeScript, SQL)? If yes — use this template and initiate SL 4 handoff. Unnecessary SL 4 escalation consumes developer capacity and delays delivery.

A-1. PRE-BUILD CHECKLIST

- ☐ Requirement defined: what operational question does this product answer?
- ☐ Audience identified: who uses it and in what context?
- ☐ Data sources identified: what datasets or Object Types are the inputs?
- ☐ Data Steward notified: coordinated with the relevant Data Steward?
- ☐ Existing products checked: is this already built somewhere that could be reused or extended?
- ☐ Naming convention selected: what will the product be named per C2DAO standards?
- ☐ Branch created: development branch created from production?

A-2. BUILD CHECKLIST

- ☐ All work done in branch (not production)
- ☐ Source datasets profiled (row counts, key distributions, NULL rates)
- ☐ NULL handling implemented in all pipelines
- ☐ Join types documented in node descriptions
- ☐ Calculated columns verified against known records
- ☐ Workshop variables follow naming convention
- ☐ All Workshop pages tested (navigation, conditional visibility, variable behavior)
- ☐ Computed columns verified for correctness
- ☐ Access controls configured to least-privilege
- ☐ Empty states handled in Workshop (no blank pages when filters return zero results)

- ☐ Action validation rules configured for all user-input parameters
- ☐ AIP Logic configurations documented and tested with sample data

A-3. REVIEW AND PROMOTION CHECKLIST

- ☐ Self-review completed — tested with production-representative data
- ☐ Downstream impact assessment completed
- ☐ Downstream owners notified of changes
- ☐ Peer review requested and completed
- ☐ Data Steward review and sign-off obtained
- ☐ Rollback plan documented
- ☐ Promotion executed in authorized window
- ☐ Post-promotion verification completed
- ☐ Downstream application owners notified of successful promotion

APPENDIX B — DESIGN PATTERNS REFERENCE

NOTE

This is the SL 3 checklist for advanced multi-page, cross-functional applications. If your application is single-page and purpose-specific, use the SL 2 checklist (SL 2, Appendix A, Section A-4, Workshop Application Checklist) instead. Before publishing any application, test it against operator workflows in TM-10, Chapter 4 (Using Workshop Applications). An application that a trained operator cannot navigate using SL 1 procedures is not ready for publication.

B-1. COMMANDER'S DASHBOARD

Purpose: Give a commander an at-a-glance operational picture with drill-down capability.

Structure: - Page 1 (Overview): Metric tiles (key readiness numbers), status bar chart, map of unit locations - Page 2 (Unit Detail): Triggered by unit selection on Page 1; shows selected unit's detail - Page 3 (Trend): Time-series charts showing readiness trend over the past 30/60/90 days

Key patterns: - Variable: `ops_selected_unit_object` — written by map or table selection, read by Page 2 - Default filter on Page 1 table: current reporting period only (avoid loading full history) - Conditional visibility on Page 2: show only when `ops_selected_unit_object` is not empty - Date range picker on Page 3: dynamic "last N days" filter driven by a variable

B-2. STATUS BOARD WITH WRITE-BACK

Purpose: Allow authorized users to update equipment or personnel status from within the application.

Structure: - Left panel: filtered table of records (user selects a record) - Right panel: detail view of selected record (appears on selection) - Bottom of right panel: Action button(s) to update status

Key patterns: - Variable: `log_selected_record_object` — written by table row click, read by detail panel and Action - Conditional visibility: right panel visible only when `log_selected_record_object` is not empty - Action configuration: pre-populate parameters from the selected object's properties where possible - Confirmation message: shows the specific record being updated and the new value before execution

B-3. READINESS ROLLUP PIPELINE

Purpose: Produce a unit-level readiness summary from an equipment dataset and maintenance records.

Pipeline structure: 1. `src_equipment_fleet` — source dataset 2. `filter_active_vehicles` — filter: status is not DISPOSED AND reporting_period is current 3. `src_maintenance_records` — source dataset 4. `filter_recent_maintenance` — filter: maintenance_date within last 30 days 5. `join_fleet_maintenance` — Left Join on vehicle_id 6. `calc_is_fmc` — calculated column: if maint_status equals "FMC" then 1 else 0 7. `agg_by_unit_category` — Group By unit_uic and equipment_category; COUNT(*) to vehicle_count, SUM(is_fmc) to fmc_count 8. `calc_readiness_rate` — calculated column: fmc_count / vehicle_count * 100 9. `calc_status_color` — calculated column: if/else producing GREEN, AMBER, or RED 10. `out_readiness_rollup` — output dataset

---

B-4. HIERARCHICAL OBJECT TYPE MODEL

Purpose: Model an operational hierarchy (Theater — Corps — Division — Brigade — Battalion — Unit).

Object Types: - `OpsTheater` — Theater-level entity - `OpsCorps` — Corps entity, linked to OpsTheater - `OpsDivision` — Division entity, linked to OpsCorps - `OpsBrigade` — Brigade entity, linked to OpsDivision - `OpsBattalion` — Battalion entity, linked to OpsBrigade - `OpsUnit` — Company/detachment entity, linked to OpsBattalion

Link pattern: Each Link Type is one-to-many (one Corps has many Divisions, etc.) **Workshop pattern:** Cascading dropdown variables drive filter from Theater down to Unit level. **Quiver pattern:** Traverse links upward and downward; aggregate metrics across the hierarchy.

B-5. DATA QUALITY DASHBOARD

Purpose: Give Data Stewards visibility into data quality issues across their domain.

Structure: - Page 1 (Summary): Counts of open issues by severity, source dataset, and age - Page 2 (Issue Queue): Table of all open issues with filters by status, severity, and dataset - Page 3 (Issue Detail): Detail view for a selected issue with resolution actions

Key patterns: - Data quality issues stored as an Object Type (`GovDataQualityIssue`) - Actions on Issue Detail page: Mark Resolved, Escalate, Add Comment - Variable: `gov_selected_issue_object` — drives Page 3 detail view - Default filter on Page 2: show only open and in-progress issues (not resolved) -

Trend chart on Page 1: shows issue count by week over rolling 90 days

GLOSSARY

Action A configured workflow on the Ontology that allows authorized users to create, modify, or delete object data through the platform UI. Actions execute validation rules before writing data.

AIP Logic The AI workflow layer of the MSS platform. Enables AI-assisted analysis, automated reasoning, and natural language query interfaces. SL 3 builders configure existing AIP Logic workflows; authoring is SL 4 scope.

API Name The machine-readable identifier for an Object Type, property, or Action. Distinct from the human-readable display name. Once published to production, the API name should not change without a coordinated downstream impact assessment.

Branching The platform mechanism for isolating development work from the production environment. Builders create a branch from production, develop in the branch, and promote to production only after review and approval.

C2DAO Command and Control Data Authoritative Organization. The USAREUR-AF governance body responsible for data product standards, naming conventions, access control policy, and production promotion approvals.

Calculated Column A derived data column whose value is computed from a formula applied to other columns. In Pipeline Builder, calculated columns are permanent and stored in the output dataset. In Workshop and Contour, they are display-only and not stored.

Cardinality In data modeling, the numerical relationship between two linked Object Types. One-to-one, one-to-many, or many-to-many. Determines how Link Types are configured in the Ontology.

CDA Portal Common Data Architecture Portal. The Army's authoritative training and reference resource for data platform work, hosted at learn-data.armydev.com.

Conditional Visibility A Workshop configuration that shows or hides a widget based on the value of an application variable. Used to create dynamic, responsive application layouts.

Contour The MSS platform's visual analytics tool. Allows builders and analysts to query, aggregate, pivot, and chart data without writing code. Produces saved analysis views that can be shared and embedded.

Data Product Owner The individual responsible for the operational accuracy, quality, and maintenance of a data product (dataset, Object Type, or application). Designated per Army CIO Memorandum (April 2024).

Data Steward The designated authority for data quality, access control, naming compliance, and governance within a data domain. Builders coordinate with Data Stewards before creating, modifying, or promoting shared resources.

Downstream Impact The effect of a change to a shared data resource on all products that depend on it — applications, pipelines, Object Types, analytics views. Assessment of downstream impact is required before any production change.

Empty State The visual condition of a Workshop widget when no data matches the current filter or no selection has been made. Advanced builders design explicit empty states rather than leaving blank panels.

Full Outer Join A join operation that returns all rows from both source datasets, with NULLs where no match exists in the other dataset.

Governance The set of policies, standards, and processes that ensure data products are accurate, accessible to authorized users, consistently named, and maintained over time. Governed by C2DAO within USAREUR-AF.

Group By A pipeline or analysis transform that collapses multiple rows into a single summary row per unique combination of grouping keys, applying aggregation functions to the non-key columns.

Inner Join A join operation that returns only rows that have a match in both source datasets. Rows without a match in either dataset are excluded from the output.

Left Join A join operation that returns all rows from the left (primary) dataset, with NULLs in right- side columns where no match exists in the right dataset.

Lineage Graph A visual representation of data flow showing the chain of sources, pipelines, and outputs that produced a given dataset or Object Type. Used for impact assessment, quality investigation, and governance audits.

Link Type An Ontology construct that defines a named relationship between two Object Types. Enables multi-object analysis in Quiver and relationship traversal in Workshop and AIP Logic.

Null Handling Explicit pipeline logic that addresses NULL (missing) values in source data — filtering, replacing with defaults, or flagging for review — before NULLs can propagate into outputs and cause analysis errors.

Object Set A filtered collection of objects from an Object Type, defined by one or more filter conditions. Used as the data source for Quiver analyses and Workshop Object Set widgets.

Object Type The fundamental ontology construct representing a class of real-world operational entities (e.g., LogVehicle, OpsUnit). Each object instance represents one specific entity.

Ontology The shared data model of the MSS platform. Defines what entities exist (Object Types), how they relate (Link Types), and what users can do to them (Actions).

Ontology Manager The platform UI for creating, configuring, and publishing Object Types, Link Types, and Actions.

Partition Pruning The platform behavior of reading only the partitions of a dataset that satisfy the filter conditions applied to the partition key. Requires that filters on the partition key are applied early in the pipeline.

Pipeline Builder The MSS platform's visual, no-code data transformation tool. Allows builders to read, join, filter, aggregate, and output datasets without writing code.

Pivot A transform that rotates a long-format dataset (one row per category per entity) into a wide- format dataset (one row per entity, one column per category). Available in both Pipeline Builder and Contour.

Primary Key The property (or combination of properties) that uniquely identifies each object instance in an Object Type. Must be unique and stable across the lifetime of the Object Type.

Production Branch The live environment that operational users depend on. No development occurs in production — all changes are promoted from reviewed development branches.

Progressive Disclosure A UX design pattern in which summary information is displayed first and detail is revealed on user selection or interaction. Reduces cognitive load and application load time.

Property A named attribute of an Object Type instance. Properties store the data about each object — its name, status, location, identifier, and other characteristics.

Quiver The MSS platform's object-centric analysis tool. Allows builders and analysts to explore Object Types, traverse Link Types, define object sets, and build multi-object dashboards.

Saved View A persistently saved Contour analysis or Quiver dashboard configuration, including all filter settings, calculated columns, and chart configurations. Enables repeatable, shareable analytical products.

UDRA Unified Data Reference Architecture. Army enterprise architecture standard (v1.1, February 2025) that defines data domains, governance roles, and architectural patterns. All USAREUR-AF data products must align to a UDRA domain.

Union A pipeline transform that combines rows from two or more datasets with compatible schemas into a single output dataset.

Variable (Workshop) A named container in a Workshop application that stores a value set by user interaction or application logic. Variables enable dynamic filtering, conditional visibility, and state management across pages.

Workshop The MSS platform's no-code application builder. Allows builders to create interactive, data-driven operational interfaces using configurable widgets without writing code.

SL 3 — Maven Smart System Advanced No-Code Builder Technical Manual Headquarters, United States Army Europe and Africa, Wiesbaden, Germany, 2026 Distribution Restriction: Distribution authorized to U.S. Government agencies and their contractors only. Other requests must be referred to Headquarters, USAREUR-AF, C2DAO, Wiesbaden, Germany.