DRAFT — UNOFFICIAL — NOT FOR OPERATIONAL USE

TECHNICAL MANUAL

SL 2

USAREUR-AF

TM-20 — MAVEN SMART SYSTEM (MSS)

Administrator Manual

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

26 MARCH 2026

DRAFT — UNOFFICIAL — NOT FOR OPERATIONAL USE

TM-20 — MAVEN SMART SYSTEM (MSS)

Forward: This manual teaches you to build applications, data pipelines, and analyses on MSS using only the graphical user interface — no coding required. **Prereqs:** SL 1, Maven Smart System Operator Manual (required). Data Literacy Technical Reference (recommended). *HQ USAREUR-AF · v1.0 · 2026 · DISTRIB: USG only · AUTH: C2DAO/UDRA v1.1*

CHAPTER 1 — INTRODUCTION

1-1. NO-CODE BUILDER MANUAL

This Technical Manual (TM) provides task-level instruction for USAREUR-AF personnel who build on the Maven Smart System (MSS) using no-code tools. It is written for all staff — officer, warrant, NCO, Civilian — who completed SL 1 and have been granted builder access. No programming background is required. If you can use a web browser and fill out forms, you can do everything in this manual.

NOTE

Before beginning SL 2 work, verify you can independently perform the following SL 1 tasks without referencing the manual: Task 2-1 (account setup and MFA); Task 3-1 (find and save resources); Task 5-1 (dataset viewing); Chapter 6 (security markings and access controls). Builders must understand operator-level data security requirements before building. If you are uncertain about any SL 1 task, review SL 1 before proceeding.

This manual covers creating and managing projects via the Compass UI; building visual ETL pipelines using Pipeline Builder (drag-and-drop, no code); connecting to data sources using pre-built UI connectors; creating Object Types and Link Types using the Ontology UI (form-based); configuring basic Actions using form-based configuration (no code); building Workshop applications with drag-and-drop widgets; publishing and sharing Workshop applications; building saved analyses in Contour; building basic dashboards in Quiver; managing project permissions and access via UI; using Foundry branching via the UI (no command line); and naming conventions and builder standards.

This manual does NOT cover Python, PySpark, or SQL transforms; code editors or code repositories; TypeScript Functions or OSDK; @incremental transform configuration; AIP Logic configuration or Agent Studio; or any task requiring writing or reading code.

Those topics are in SL 3, Advanced Builder. If a task requires writing code, stop and contact your team's data engineer.

1-2. THINK BEFORE YOU BUILD — THE BUILDER'S DESIGN FRAMEWORK

BLUF: The most common builder failure is building the wrong thing correctly. Answer these three questions before you open Pipeline Builder or Workshop. Spend time here before spending time on the platform.

Every build task starts with three questions. Answer them in order — do not skip ahead.

Question 1: Who is my user and what decision are they making?

The application or pipeline you build exists to help someone make a decision or take an action. If you cannot name that person and that decision before you start, you are not ready to build.

Examples of clear answers: - "The S4 NCO in Graf needs to see which vehicles are deadlined, updated daily, so they can prioritize maintenance scheduling." - "The battalion XO needs to see personnel readiness by company so they can report up to brigade before the Monday briefing."

Examples of answers that are too vague: - "The S4 needs a readiness dashboard." (What kind of readiness? What does 'ready' mean? What action follows?) - "G2 wants to see data about units." (Which units? What about them? For what purpose?)

If you have a vague answer, get a clearer requirement before you build. Building to a vague requirement produces something that looks complete but does not actually help anyone.

Question 2: What data do I need, and where does it come from?

Map the pipeline before you build it:

```
DECISION REQUIREMENT          DATA ELEMENTS NEEDED          SOURCE SYSTEM
"Which vehicles are   →   Equipment ID, status,    →   GCSS-A (via existing
 deadlined by fleet?"       class, location, date        ingestion pipeline)
```

Verify the data exists and is already in MSS before you design anything that depends on it. If the data does not exist, you have an ingestion problem — not a Workshop problem. Ingestion decisions require Data Steward coordination.

Question 3: Where does my work fit in the three-phase workflow?

Use the three-phase workflow from paragraph 1-4 as a design checklist:

Phase	Question to ask
Phase 1 — Pipeline Builder	Is the data I need already ingested? If not, can Pipeline Builder connect to the source?
Phase 2 — Ontology	Is there an Object Type that represents the thing I'm building around? If not, does one need to be created (Data Steward coordination required)?

Phase	Question to ask
Phase 3 — Workshop	What widgets best present the decision my user needs to make? Does the user need to take an Action — submit a form, trigger a workflow — or just view information?

Build bottom-up. Do not start building a Workshop application until you have confirmed the data layer beneath it is solid. A beautifully designed Workshop app built on a broken pipeline is useless.

CAUTION

A common mistake is building at Phase 3 (Workshop) while assuming the Phase 1 and Phase 2 problems will work themselves out. They will not. Verify your data exists, is clean, and is modeled correctly in the Ontology before you build the application that depends on it.

The builder's minimum viable check before starting any task:

- ☐ I can name the user and the decision they need to make.
- ☐ I know what data is needed and have confirmed it exists in MSS (or have a plan to get it there).
- ☐ I know which Object Type(s) my application will be built on.
- ☐ I have looked at the existing application (if any) before building something new.
- ☐ I have talked to the intended user — not just their supervisor — about what they need.

If any box is unchecked, do not build. Resolve it first.

1-3. USAREUR-AF MISSION CONTEXT AND THE BUILDER'S ROLE

United States Army Europe and Africa (USAREUR-AF) is the Army Service Component Command (ASCC) to United States European Command (USEUCOM) and United States Africa Command (USAFRICOM), responsible for theater land operations across the European and African Areas of Responsibility (AOR) and integration with NATO Allied command structures. Major subordinate commands — III Corps, V Corps (forward deployed, Poland), 21st Theater Sustainment Command (TSC), 7th Army Training Command (ATC), 10th AAMDC, 56th MDC-E, and SETAF-AF — each generate and consume data that flows through MSS.

As a builder, the tools you create directly affect readiness visibility and operational decision-making across this formation. A Workshop application you build may display unit status to a V Corps G3 in Poznan or track logistics readiness for a 21st TSC officer in Kaiserslautern. A pipeline you configure may feed the data behind a theater-level briefing. Understand the operational weight of what you are building before you begin.

Per ADP 3-13, information is combat power. Builders are one of the primary means by which raw data becomes operationally useful information for commanders. This manual supports data operations governed by AR 25-1, *Army Information Technology*, which establishes policy for Army data management. The original VAUTI data quality framework (5 dimensions, AR 25-1, 2019) has been superseded by VAULTIS (7 dimensions, DoD Data Strategy 2020) and extended to VAULTIS-A (8 dimensions, DDOF Playbook v2.2, December 2025). VAULTIS-A dimensions — Visible, Accessible, Understandable, Linked, Trusted, Interoperable, Secure, Auditable — with an 85% minimum weighted average across all 8 dimensions constituting the DDOF Phase 3 quality gate. Proponent: T2COM C2DAO / HQDA CIO/G-6 / SAIS-ADD.

NOTE

As a builder, understand the operator's perspective before you build. Refer to TM-10, Chapter 4 (Using Workshop Applications) to see how operators use the applications you create. Refer to TM-10, Chapter 5 (Working with Data) to understand what operators expect in terms of data quality and currency. Build with the operator experience in mind at all times.

1-4. THE BUILDER'S THREE-PHASE WORKFLOW

SL 2 builders work in three sequential phases. Complete each phase before beginning the next.

Phase 1 — Get and clean the data (Pipeline Builder). Ingest data from a source, clean it, and produce a curated output dataset. This is the foundation. Without clean, reliable data, nothing downstream works.

Phase 2 — Organize it so the system understands it (Ontology). Create Object Types backed by your curated data, configure Link Types between related objects, and define Actions users can perform. This is the semantic layer that turns a data table into meaningful, connected objects.

Phase 3 — Build the application users see (Workshop). Assemble widgets — tables, charts, filters, forms — into an application that operators use daily. Workshop reads from the Ontology. It does not read directly from datasets.

```
Phase 1: Pipeline Builder  →  Phase 2: Ontology  →  Phase 3: Workshop
(clean, curated data)          (meaningful objects)    (user-facing app)
```

Do not skip. Complete Phase 1 before Phase 2. Complete Phase 2 before Phase 3. Building Phase 3 before Phase 1 and Phase 2 are solid produces an application with no reliable data behind it.

For the full platform architecture (five-layer stack diagram and layer-to-activity mapping), see Appendix C of the USAREUR-AF Data Literacy Technical Reference.

1-5. PREREQUISITES AND SL 2 SCOPE BOUNDARY

Access requirements:

- ☐ SL 1 (Maven User) completed
- ☐ Builder access request submitted through chain of command and approved
- ☐ Editor role granted on your team's project folder in Compass
- ☐ Editor role granted on the Ontology branch for your team
- ☐ Workshop Builder permission granted

Orientation requirements (complete before creating any resource):

- ☐ Navigated to your team's project folder in Compass
- ☐ Opened and reviewed at least one existing Pipeline Builder pipeline
- ☐ Previewed the output dataset of that pipeline
- ☐ Found the Object Type backed by that dataset in Ontology Manager
- ☐ Opened and used (as an end user) the Workshop application built on that Object Type

Do not begin building until you understand the existing system. Adding to something you do not understand creates problems that are difficult to untangle.

SL 2 Scope Boundary:

The table below identifies requirements that exceed SL 2 scope. If your requirement falls in the left column, escalate — do not attempt it as a SL 2 build.

IF the requirement needs this...	THEN escalate to...
Multi-step Actions with conditional routing, sequential submission steps, approval chains, or multi-record writes	TM-30, Chapter 4, Task 4-3
Multi-source joins (3+ sources), fan-out handling, post-join deduplication, or union transforms	TM-30, Chapter 3
AIP Logic configuration or Agent Studio integration	SL 3
Multi-page Workshop applications with conditional navigation between pages	TM-30, Chapter 2
@incremental pipeline patterns	TM-30, Chapter 3
Many-to-many Link Types with junction dataset logic	TM-30, Chapter 4
Python, PySpark, or SQL code transforms	SL 4L (Software Engineer)
TypeScript Functions or OSDK	SL 4L (Software Engineer)

IF the requirement needs this...	THEN escalate to...
Machine learning model integration	SL 4M (ML Engineer)

NOTE

Join scope boundary — SL 2 covers joining two datasets on a single well-known key (1:1 or 1:M relationships). This is the standard equipment-to-unit, unit-to-event join pattern. If your requirement involves three or more sources, fan-out handling after a join, group-by aggregations across sources, or union transforms combining differently-shaped datasets, that is SL 3 scope — escalate before attempting it.

1-6. GOVERNANCE AND ESCALATION CHAIN

All builder activity on MSS is governed by the USAREUR-AF Data Governance chain. Know this chain before you build anything.

```
ARMY CIO
  |  Army Data Plan (2022); CIO Data Stewardship Policy (Apr 2024)
  v
USAREUR-AF C2DAO
  |  Theater-level data governance authority
  |  Approves new data sources, ontology changes, access policies
  v
FUNCTIONAL DATA MANAGER
  |  Day-to-day oversight of a specific data domain
  v
UNIT DATA STEWARD
  |  Your unit's accountable data official
  |  First point of contact for any data question
  v
BUILDER (YOU)
     Build within approved scope on approved data
```

Key policy references:

Document	Governing Authority	Relevance to SL 2 Builders
USAREUR-AF Data Governance SOP	C2DAO	USAREUR-AF-specific naming, access, and approval processes

CAUTION

Before ingesting any new data source, coordinate with your unit Data Steward and the USAREUR-AF C2DAO. Unauthorized ingestion of operational data — even from Army systems — may violate Army CIO policy and create compliance findings.

1-7. REFERENCES

- USAREUR-AF C2DAO Data Governance SOP (current version)
- Palantir Foundry Product Documentation (in-platform Help)
- C2DAO: ontology design standards, doctrine-aligned Object Type patterns, data modeling guidance

DoD and Army Strategic References:

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

- **DoD Data Strategy (October 2020)** — Foundation for VAULTIS principles (Visible, Accessible, Understandable, Linked, Trusted, Interoperable, Secure). VAUTI (AR 25-1, 2019) is superseded. Current standard: VAULTIS-A (8 dimensions, DDOF Playbook v2.2, Dec 2025), adding Auditable. 85% weighted average across all 8 dimensions = DDOF Phase 3 quality gate.
- **Army Data Plan (2022)** — 11 strategic objectives for Army data transformation
- **Army Cloud Plan (2022)** — Zero Trust, secure development, data-driven decisions
- **Army CIO Data Stewardship Policy (April 2, 2024)** — Data stewardship hierarchy; data chain of responsibility

1-8. ADVANCEMENT FROM SL 2 — NEXT STEPS

After completing SL 2, builders may advance to more specialized tracks based on their role and duties. Two advancement paths are available.

Next Step — Advanced Builder (required for all SL 4 tracks):

All personnel who complete SL 2 proceed to SL 3 (Advanced Builder) before enrolling in any SL 4 track. SL 3 is a hard prerequisite — no waivers — for both WFF tracks (SL 4A–F) and Specialist tracks (SL 4G–O).

WFF Tracks (SL 4A–F) — available after SL 3:

Complete the WFF track aligned to your functional area. These tracks focus on MSS product operation within your warfighting function — no code-level skills required. Prerequisite: SL 3 (required). Duration: 3 days each.

Track	Title	For Personnel
SL 4A	Intelligence WFF	G2 / S2 staff
SL 4B	Fires WFF	Fire support personnel
SL 4C	Movement & Maneuver WFF	G3 / S3 staff
SL 4D	Sustainment WFF	G4 / S4 / logistics staff
SL 4E	Protection WFF	Protection officers and NCOs
SL 4F	Mission Command WFF	G6 / S6 and command staff

Specialist Tracks (SL 4G–O) — available after SL 3:

Specialist tracks require code-level skills in addition to SL 3. These tracks are NOT reachable from SL 2 directly. Prerequisite: SL 3 (required).

Track	Title	For Personnel	Duration
SL 4G	ORSA	Operational research analysts	5 days
SL 4H	AI Engineer	AIP Logic / AI workflow developers	5 days
SL 4M	ML Engineer	Machine learning pipeline developers	5 days
SL 4J	Program Manager	Data program managers	4 days
SL 4K	Knowledge Manager	Knowledge management specialists	4 days
SL 4L	Software Engineer	Python / TypeScript / OSDK developers	5 days
SL 4N	UI/UX Designer	User interface and experience designers	5 days
SL 4O	Platform Engineer	Platform infrastructure and deployment specialists	5 days

Train-the-Trainer Track — available after SL 2:

T3-F (MSC Force Multiplier) is a half-day Unit Data Trainer certification. Prereq: SL 2 Go + commander nomination. Authorizes SL 1 delivery and SL 1 exam proctoring. T3-F does NOT require SL 3.

NOTE

If you are unsure which path applies to your billet, consult your unit data steward or the USAREUR-AF C2DAO training coordinator.

CHAPTER 2 — PROJECT SETUP AND MANAGEMENT

2-1. MSS RESOURCE HIERARCHY

Before creating anything, understand how MSS organizes resources in Compass.

Container	What It Holds	Managed By
Project	Top-level grouping for one functional area	Team lead / C2DAO approval
Folder	Subdirectory within a project	Builder
Dataset	A managed table produced by a pipeline	Builder (via Pipeline Builder)
Pipeline	A visual ETL flow (Pipeline Builder)	Builder
Ontology	Shared semantic layer — Object Types, Links, Actions	Builder (via Ontology Manager)
Workshop App	A published application for end users	Builder

Your team's project already exists. Do not create a new top-level project without authorization from your unit Data Steward and C2DAO approval.

2-2. STANDARD FOLDER STRUCTURE

Within an authorized project, create folders following this standard structure.

```
/[AOR]-[FUNCTION]/
+-- raw/              data as it arrives from source systems
+-- staging/          cleaned, validated, intermediate data
+-- curated/          publication-ready data, Ontology-backed
+-- pipelines/        Pipeline Builder definitions
+-- applications/     Workshop app definitions
```

Example — EUCOM SITREP project:

```
/USAREUR-AF-SITREP/
+-- raw/
|   +-- sitrep_feed_raw
+-- staging/
|   +-- sitrep_feed_staging
+-- curated/
```

```
|   +-- sitrep_feed_curated
+-- pipelines/
|   +-- sitrep-ingestion-pipeline
+-- applications/
    +-- EUCOM SITREP Dashboard
```

NOTE

Dataset paths are permanent. You cannot rename a dataset path without breaking all downstream pipelines and Ontology objects. Plan folder and dataset names before creating any resource. When in doubt, ask your team lead.

NOTE

The folder structure you create is what operators navigate using TM-10, Chapter 3 (Navigating the Platform), specifically Task 3-1 (Find and Save Resources). Poor folder structure and unclear naming conventions create confusion for operators downstream. Test your folder organization by navigating it as an operator would before finalizing.

2-3. NAMING CONVENTIONS

All resources must follow USAREUR-AF naming conventions. Non-compliant resources will be renamed or removed by the team lead.

Resource Type	Convention	Example
Dataset — raw tier	`[source]_raw`	`sitrep_feed_raw`, `gcss_a_maint_raw`
Dataset — staging tier	`[source]_staging`	`sitrep_feed_staging`
Dataset — curated tier	`[source]_curated`	`sitrep_feed_curated`
Pipeline Builder pipeline	`[function]-pipeline`	`sitrep-ingestion-pipeline`
Object Type	PascalCase, singular noun	`UnitStatus`, `SoldierReadiness`, `MaintenanceRecord`
Link Type	camelCase verb phrase	`assignedTo`, `reportedBy`, `locatedAt`
Workshop App	Plain English, unit-appropriate	`EUCOM SITREP Dashboard`, `Unit Readiness Tracker`

Resource Type	Convention	Example
Ontology branch	`dev-[feature]` or `dev-[lastname]`	`dev-sitrep`, `dev-rodriguez`
Project folder	`[AOR]-[FUNCTION]` all caps	`USAREUR-AF-SITREP`, `VCORPS-READINESS`

CAUTION

Never use PII, classified nicknames, operational codenames, or system passwords in any resource name, description, or label. Resource names and descriptions are visible to all users with project access and appear in audit logs.

TASK 2-4: USE SOLUTION DESIGNER TO PLAN A BUILD PROJECT

TASK: Use Solution Designer to create a visual diagram of a planned build project before writing any pipelines, creating any Object Types, or building any Workshop applications.

CONDITIONS: Builder has received a requirement and is preparing to build. The requirement has been confirmed with the user and the data steward. Builder has completed the three-question design framework (paragraph 1-2) before opening Solution Designer.

STANDARDS: Builder produces a Solution Designer diagram that accurately represents the planned data flow — from source through pipeline to dataset, through Object Type, to Workshop application — annotated with design decisions and data flow direction, and shared with the data steward for review before any build work begins.

EQUIPMENT: MSS account with Editor role; Solution Designer access within the project.

DURATION: 1–2 hours.

PROCEDURE:

1. In Compass, navigate to your team's project folder.
2. Click **New** and look for **Solution Designer** or **Diagram** in the resource creation menu. If Solution Designer is not available in the creation menu, search for "Solution Designer" in the platform search bar and open it from there.
3. Click **New Diagram**. Name the diagram using the naming convention `[project-name]_solution-design_v[version]` (e.g., `vcorps-readiness_solution-design_v1`).

Step 1 — Add nodes for your planned build components:

1. From the node palette (left panel or drag-to-canvas menu), add nodes for each component you plan to build:

2. **Source Dataset** — the raw or existing dataset you will ingest or connect to
3. **Pipeline** — the Pipeline Builder pipeline that will clean or transform the data
4. **Output Dataset** — the curated dataset the pipeline produces
5. **Object Type** — the Ontology Object Type backed by your curated dataset
6. **Workshop Application** — the application operators will use
7. If your build includes multiple sources, add a node for each source. If it includes Link Types between Object Types, add nodes for each Object Type.

Step 2 — Connect nodes to show data flow:

1. Draw edges (arrows) between nodes to show the direction of data flow. The diagram should read left to right: source → pipeline → dataset → Object Type → Workshop App.
2. Label each edge to describe what data flows across it. For example: "Vehicle status rows" or "Joined and deduplicated records."

Step 3 — Annotate design decisions:

1. Add annotation nodes or comment boxes to capture key design decisions:
2. What is the join key between two sources?
3. What is the primary key of the Object Type?
4. Which Actions will be configured on the Object Type?
5. Who is the intended user of the Workshop application?
6. Note any assumptions: "Assumes GCSS-A feed is already ingested by the logistics pipeline team."
7. Note any open questions: "Need to confirm with data steward whether G4 has write-back authorization."

Step 4 — Share for review:

1. Save the diagram.
2. In the diagram settings or sharing panel, share the diagram with your data steward and team lead.
3. Do not begin building pipelines or Object Types until the diagram has been reviewed and approved. A 30-minute diagram review prevents hours of rework.

NOTE

Solution Designer is a planning tool — it does not generate any pipelines, Object Types, or applications automatically. It is a shared visual reference. Keep the diagram updated as the build evolves; a diagram that accurately reflects the final build is useful for future maintainers.

TASK 2-5: BUILD A PROJECT ROADMAP USING FORWARD AND BACKWARD PLANNING

TASK: Before beginning any build, produce a documented project roadmap using forward and/or backward planning to sequence the build correctly and ensure the final product addresses the commander's requirement.

CONDITIONS: Builder has received a confirmed requirement and has completed Task 2-4 (Solution Designer diagram). Builder has confirmed data availability with the data steward.

STANDARDS: Builder produces a documented roadmap — either as annotations in the Solution Designer diagram, a written outline, or both — that sequences the build phases in the correct order, identifies dependencies, and scopes the build to what the data actually supports. Roadmap is reviewed and agreed upon with the data steward before build begins.

EQUIPMENT: MSS account; Solution Designer diagram from Task 2-4; confirmed data availability.

DURATION: 30–60 minutes.

PROCEDURE:

When to use forward planning:

Forward planning starts from the data you have and designs toward a product the data can support. Use forward planning when the data source is well-understood and you are determining what products are feasible.

1. Start at the source dataset. What columns does it have? What is the grain (one row per what)?
2. Ask: given this data, what can I build that is operationally useful?
3. Design the pipeline, Object Type, and application around the data's actual content.
4. Document: "The GCSS-A vehicle feed supports a readiness dashboard by fleet and unit. It does not support maintenance history trend analysis — that requires the maintenance event feed."

When to use backward planning:

Backward planning starts from the commander's requirement and works backward to determine what data, pipelines, and objects are needed. Use backward planning when the requirement is clear and you need to determine whether it is buildable.

1. Start at the requirement: "The S4 needs a daily view of deadlined vehicles by company within the brigade."
2. Ask: what data is needed to produce this? (Vehicle ID, status, deadline reason, unit assignment.)
3. Ask: does that data exist in MSS? (Check with the data steward.)
4. Ask: is it in a format that supports this view? (One row per vehicle, with unit UIC.)
5. Work backward through the pipeline and source to confirm the build is feasible.

6. If the data does not exist or does not support the requirement: stop and report to the data steward before continuing. Do not build a workaround without authorization.

Documenting the roadmap:

1. Sequence the build into phases using the three-phase workflow (paragraph 1-4):
2. Phase 1 — Pipeline: which pipeline(s) must be built first?
3. Phase 2 — Ontology: which Object Type(s), Link Types, and Actions?
4. Phase 3 — Workshop: which application pages, widgets, and filters?
5. Identify dependencies: "Phase 2 cannot begin until Phase 1 pipeline produces a clean dataset."
6. Identify scope limits: "This build covers vehicle status only. Personnel readiness is out of scope for this iteration."
7. Share the roadmap with the data steward and the intended user's representative for confirmation.

> **CAUTION**
>
> Build to the requirement, not to the maximum of what the data could theoretically support. A scoped build that solves the user's actual problem is more valuable than a comprehensive build that takes three times as long and is never fully used. Agree on scope before building — changes to scope mid-build are the leading cause of builder rework.

TASK 2-1. CREATE A PROJECT

TASK: Create a new project in Compass for an authorized functional area.

CONDITIONS: Builder access granted; no existing project for your functional area; unit Data Steward and C2DAO have authorized project creation.

STANDARDS: The builder will create a project in Compass with the correct name and folder structure, assign appropriate initial permissions, and confirm the project is visible to authorized team members before creating any resources inside it.

EQUIPMENT: MSS account with Owner or Manager role; Compass access; C2DAO project authorization memo.

DURATION: 30--45 minutes.

PROCEDURE:

1. In Compass, click **New** (top right) and select **Project**.
2. Enter the project name following the `[AOR]-[FUNCTION]` convention (e.g., `VCORPS-READINESS`).
3. Enter a description: include the owning unit, functional area, data steward POC, and creation date.
4. Click **Create**.
5. Inside the new project, create subfolders: `raw`, `staging`, `curated`, `pipelines`, `applications`.

6. To create a folder: click **New** then **Folder**, enter name, click **Create**.
7. After folder structure is complete, set permissions (Task 2-3) before adding any data.

> **NOTE**
>
> Do not create datasets or pipelines before folder structure and permissions are set. Fixing structure after data is created requires migrating resources, which breaks downstream references.

TASK 2-2. ADD TEAM MEMBERS TO A PROJECT

TASK: Add an authorized team member to an existing project with the correct role.

CONDITIONS: Builder has Owner or Manager role on the project; team member has an active MSS account; team lead has authorized access.

STANDARDS: The builder will add the team member with the correct role. The new member can access the project at the appropriate permission level within 5 minutes. No member is assigned a role higher than required for their duties.

EQUIPMENT: MSS account with Owner or Manager role on the project; approved access list from Data Steward.

DURATION: 15--20 minutes.

PROCEDURE:

1. Open the project in Compass.
2. Click the **Settings** icon (gear) in the upper right of the project view.
3. Select **Access and Sharing**.
4. Click **Add Member**.
5. Search for the team member by name or email.
6. Select the appropriate role from the dropdown (see role table below).
7. Click **Add**.
8. Verify the member appears in the member list with the correct role.
9. Notify the member that access has been granted.

Role Reference Table:

Role	Can Do	Use For
Viewer	Read datasets, open Workshop apps	End users, consumers
Editor	Modify pipelines, datasets, Workshop apps	SL 2 builders
Owner	All Editor rights plus manage members, delete resources	Team leads

NOTE

Assign the minimum role required. If a team member only consumes data in Workshop apps, assign Viewer — not Editor. Follow least-privilege principles per Army CIO policy.

CAUTION

Do not assign Owner role without authorization from your Data Steward. Owner access allows deletion of shared datasets and modification of permissions for all project members.

TASK 2-3. SET PROJECT PERMISSIONS AND ACCESS CONTROLS

TASK: Configure project permissions to match the approved access list.

CONDITIONS: Builder has Owner role on the project; unit Data Steward has provided an approved access list; classification and handling requirements are known.

STANDARDS: The builder will configure project permissions so that only authorized personnel have access, roles correctly match duties, and the Data Steward has reviewed and approved the configuration before any data is added.

EQUIPMENT: MSS account with Owner role; approved access list from Data Steward.

DURATION: 20--30 minutes.

PROCEDURE:

1. Open the project in Compass, click **Settings**, then **Access and Sharing**.
2. Review the current member list against the approved access list from the Data Steward.
3. Remove any members not on the approved list: click the three-dot menu next to their name then **Remove**.
4. Add authorized members not yet listed (see Task 2-2).
5. Verify each member's role against their duty position and adjust as needed.
6. Set the **Project Visibility**:
7. **Private** — access by invitation only (default; use for all operational data projects)
8. **Organization** — visible to all MSS users (use only for unclassified reference/training projects)
9. Click **Save**.
10. Send the finalized access list to your Data Steward for confirmation.

CAUTION

Do not set any operational data project to Organization visibility without explicit C2DAO approval. This makes all datasets in the project readable by all MSS users.

CHAPTER 3 — DATA INGESTION WITH PIPELINE BUILDER

3-1. WHAT PIPELINE BUILDER DOES

Pipeline Builder is MSS's visual, no-code ETL (Extract, Transform, Load) tool. Using a drag-and-drop interface, you build pipelines that pull data from a source, clean or reshape it, and write the result to a destination dataset — no code required.

Where Pipeline Builder fits in the three-phase workflow: Phase 1. The output of every Pipeline Builder pipeline is a dataset that feeds Phase 2 (Ontology) and Phase 3 (Analytics/Workshop).

What Pipeline Builder can do without code:

- Connect to pre-configured data sources (files, shared datasets, approved connectors)
- Filter rows by value, date range, or condition
- Select, rename, and reorder columns
- Join two datasets on a shared key
- Aggregate data (count, sum, average, group by)
- Append (union) multiple datasets
- Deduplicate rows
- Schedule automatic refreshes

What Pipeline Builder cannot do (requires SL 3/code):

- Complex multi-step transformations
- Custom business logic
- Machine learning or statistical computation
- Writeback to external systems

NOTE

The pipelines you build in Pipeline Builder become the data sources that operators access via TM-10, Task 5-1 (View and Read a Dataset) and in Workshop applications. Pipeline failures directly impact operators' ability to access operational data. Refer to TM-10, Chapter 7 (Troubleshooting and Support) to understand the operator experience when a pipeline fails — build your monitoring accordingly.

3-2. PIPELINE BUILDER INTERFACE OVERVIEW

UI Area	Location	Purpose
Canvas	Center	Where you build — drag nodes here and connect them
Node Library	Left panel	Available data sources, transforms, and outputs
Node Properties	Right panel	Configuration for the selected node
Preview Pane	Bottom panel	Live data preview — shows first 100 rows of selected node output
Build Log	Bottom panel (tab)	Build status, errors, warnings
Schedule	Top toolbar	Configure automatic refresh
Branch	Top toolbar	Shows current branch; switch branch before editing

Always work on a branch, not on the main/production pipeline. See Chapter 7 for branching.

TASK 3-1: CREATE A PIPELINE BUILDER PIPELINE

TASK: Create a new Pipeline Builder pipeline in the correct project folder.

CONDITIONS: Builder has Editor access on the project; source dataset or connector is available; destination folder exists; working on a development branch (not main).

STANDARDS: The builder will create a Pipeline Builder pipeline in the correct project folder, connect a valid source, add at least one transform node, connect an output dataset, and confirm the pipeline builds successfully (green checkmark, no errors) before requesting a merge.

EQUIPMENT: MSS account with Editor role; Pipeline Builder access; development branch active.

DURATION: 30--60 minutes.

PROCEDURE:

1. In Compass, navigate to your project's `pipelines/` folder.
2. Click **New**, then **Pipeline**, then **Pipeline Builder**.
3. Name the pipeline following the convention `[source]-pipeline` (e.g., `sitrep-ingestion-pipeline`).
4. The Pipeline Builder canvas opens. Confirm the branch shown in the top toolbar is your development branch — not `main`.
5. In the **Node Library** (left panel), expand **Sources**.

6. Drag the appropriate source node onto the canvas (see Task 3-2 for connecting sources).
7. Add transform nodes as needed (see Task 3-3).
8. Drag an **Output Dataset** node from the Node Library onto the canvas.
9. Connect nodes by clicking the output port of one node and dragging to the input port of the next.
10. Click the Output Dataset node. In the right panel, navigate to your project's appropriate folder and name the dataset.
11. Click **Build** (top toolbar). Wait for the build to complete.
12. If the build succeeds (green checkmark): click the Output node, then **Preview** (bottom panel), and verify the data looks correct.
13. If the build fails: read the error in the Build Log, fix the node configuration, and rebuild.

> **NOTE**
>
> A successful build means the pipeline ran without errors. It does not mean the data is correct. Always preview the output and spot-check values before declaring the pipeline complete.

BUILDER VIGNETTE — S4 Equipment Readiness Pipeline

SSG Kim is the data NCO for 3rd ABCT at Rose Barracks. Her S4 officer needs a Workshop application showing which vehicles are deadlined and which are FMC, updated daily from GCSS-A.

SSG Kim follows the three-question framework: *Who needs this?* — the S4 officer. *What decision does it support?* — maintenance prioritization and readiness reporting. *What data is required?* — vehicle status and deadline reason from GCSS-A.

She builds a Pipeline Builder pipeline that ingests the GCSS-A vehicle status feed, filters to her brigade's UICs, renames columns to plain English (e.g., `veh_status_cd` → `Vehicle Status`), and outputs a clean dataset. She previews the output and spot-checks five rows: vehicle IDs are present, status values are either "FMC" or "NMC", deadline reasons are populated for NMC vehicles. The pipeline builds successfully (green checkmark). She schedules it to run at 0300 daily.

In Chapter 4, she creates a `VehicleStatus` Object Type backed by her clean dataset, with `Vehicle_ID` as the primary key. In Chapter 5, she builds a Workshop application with a filter for unit and a table showing vehicle ID, status, and deadline reason. She tests on her development branch, peer reviews with her data steward, and promotes to production.

The S4 officer now has a live readiness view every morning before PT formation.

TASK 3-2: CONNECT AND PREVIEW A DATA SOURCE

TASK: Connect a data source node in Pipeline Builder and verify the incoming data schema and row count.

CONDITIONS: Builder is in Pipeline Builder on a development branch; a pre-configured connector or source dataset exists; Data Steward has approved ingestion of this source.

STANDARDS: The builder will connect the data source, preview the incoming data to confirm schema and row count, and document the source name, expected row count, and refresh frequency in the pipeline's description field before proceeding.

EQUIPMENT: MSS account with Editor role; Pipeline Builder access; authorized connector or source dataset path; Data Steward approval documented.

DURATION: 20--40 minutes.

PROCEDURE:

Option A — Connect to an existing MSS dataset:

1. From the Node Library, drag **Dataset** (under Sources) onto the canvas.
2. In the right panel, click **Select Dataset**.
3. Browse Compass to the source dataset or search by name.
4. Click the dataset then **Confirm**.
5. Click the source node then **Preview** (bottom panel). Verify schema and row count.

Option B — Connect via a pre-built connector:

1. From the Node Library, drag **Connector** onto the canvas.
2. In the right panel, click **Select Connector**.
3. From the list of authorized connectors, select the appropriate connector (e.g., GCSS-A feed, SharePoint file, SFTP source).
4. Configure the connection parameters (folder path, file pattern, authentication — use values provided by your Data Steward; do not enter credentials from memory).
5. Click **Test Connection**. Verify the connection succeeds.
6. Click the connector node then **Preview**. Verify the incoming data schema.

CAUTION

Before connecting any new data source, coordinate with your unit Data Steward and the USAREUR-AF C2DAO. Do not connect to a source that has not been explicitly approved for ingestion into MSS. This applies even if you personally have access to the source system.

NOTE

If the connector you need is not on the authorized list, submit a request to the C2DAO through your Data Steward. Do not attempt to work around missing connectors by manually uploading data files without approval.

TASK 3-3: ADD AND CONFIGURE TRANSFORM NODES

TASK: Add transform nodes to a pipeline to filter, reshape, or join data.

CONDITIONS: Builder is in Pipeline Builder on a development branch; a source node is connected; need to filter, reshape, or join data.

STANDARDS: The builder will add the correct transform nodes for the required operation, configure each node's properties accurately, and verify the output in the preview pane before connecting to the output dataset.

EQUIPMENT: MSS account with Editor role; Pipeline Builder access; source node connected.

DURATION: 30--60 minutes.

PROCEDURE — Filter rows:

1. From Node Library, drag **Filter** onto the canvas between your source and output.
2. Connect: source to Filter, Filter to output.
3. In the right panel, click **Add Condition**.
4. Select the column to filter on from the dropdown.
5. Select the operator (equals, contains, greater than, is not null, etc.).
6. Enter the filter value.
7. For multiple conditions, click **Add Condition** again and select AND/OR logic.
8. Click the Filter node then **Preview** to confirm filtered row count.

PROCEDURE — Select and rename columns:

1. Drag **Select Columns** onto the canvas.
2. In the right panel, check/uncheck columns to include or exclude.
3. To rename a column: click the column name in the right panel and edit the name.
4. Preview to confirm column list is correct.

PROCEDURE — Join two datasets:

1. Drag **Join** onto the canvas.
2. Connect the first (left) dataset to the left input port; the second (right) dataset to the right input port.

3. In the right panel, select the **Join Type** (Inner, Left, Right, Full Outer).
4. Under **Join Keys**, select the matching column from the left dataset and from the right dataset.
5. Under **Output Columns**, select which columns to include from each side.
6. Preview to confirm joined row count and verify no unexpected row multiplication.

> **NOTE**
>
> A join that multiplies rows unexpectedly usually means the join key is not unique on one side. Check for duplicate keys in your source data before troubleshooting the join configuration.

PROCEDURE — Aggregate (group by/summarize):

1. Drag **Aggregate** onto the canvas.
2. Under **Group By**, select the columns to group on (e.g., `unit_id`, `report_date`).
3. Under **Aggregations**, click **Add Aggregation**. Select the column and function (Count, Sum, Average, Min, Max).
4. Name the output column for each aggregation.
5. Preview to confirm row count is reduced (one row per unique group).

TASK 3-4: CONFIGURE PIPELINE OUTPUT AND SCHEDULE

TASK: Configure the pipeline output dataset and automatic refresh schedule.

CONDITIONS: Pipeline builds and produces correct output; builder has authorization from Data Steward to schedule automatic runs; the source data updates on a known schedule.

STANDARDS: The builder will configure a schedule that matches the source data refresh cadence, set appropriate alerting, and confirm the first scheduled run completes successfully.

EQUIPMENT: MSS account with Editor role; successfully building pipeline; Data Steward schedule authorization.

DURATION: 20--30 minutes.

PROCEDURE:

1. In the pipeline (after successful build), click **Schedule** in the top toolbar.
2. Click **Add Schedule**.
3. Select the frequency:
4. **Hourly** — for near-real-time feeds (use sparingly; confirm with Data Steward)
5. **Daily** — most common for SITREP, readiness, and logistics feeds

6. **Weekly** — for slower-changing data
7. Set the start time. For daily feeds supporting morning briefings, schedule for 0300-0400 local to allow build time.
8. Under **On Failure**, configure email notification: enter your email and your team lead's email.
9. Click **Save Schedule**.
10. To manually trigger the first run: click **Build Now** and monitor the Build Log.
11. After the first run: preview the output dataset to confirm correct results.

NOTE

Schedule times are in UTC. USAREUR-AF is UTC+1 (CET) or UTC+2 (CEST in summer). A 0300 CET scheduled run should be entered as 0200 UTC in winter, 0100 UTC in summer.

CAUTION

Do not schedule pipelines to run more frequently than the source data actually updates. Excessive builds consume platform resources and may cause rate-limiting on source connectors.

TASK 3-5: MONITOR A PIPELINE AND RESPOND TO BUILD FAILURES

TASK: Monitor a scheduled pipeline and diagnose and resolve build failures.

CONDITIONS: Pipeline is scheduled and running; builder has received a failure alert or is performing routine monitoring.

STANDARDS: The builder will identify the failing node and error cause within 15 minutes of notification, take corrective action or escalate appropriately within 1 hour, and document the failure and resolution in the pipeline's description field.

EQUIPMENT: MSS account with Editor role; pipeline access; Build Log visible.

DURATION: 30--90 minutes (depending on issue complexity).

PROCEDURE:

1. In Compass, open the pipeline.
2. Click the **Build Log** tab (bottom panel).
3. Find the failed build (red icon) — note the timestamp.
4. Click the failed build entry to expand the error details.
5. Identify the failing node: Pipeline Builder highlights failing nodes in red on the canvas.
6. Click the failing node and read the error message in the right panel.

Common failure causes and actions:

Error Type	Likely Cause	Action
Source dataset not found	Dataset was renamed or moved	Update the source node path
Column not found	Source schema changed	Update column names in affected transform nodes
Connection timeout	Source system unavailable	Check source system status; retry manually; alert Data Steward if persistent
Authentication failure	Connector credentials expired	Contact Data Steward — do not attempt to update credentials yourself
Row count zero	Source has no data for this run	Investigate source; may be expected for some time windows
Build timed out	Pipeline too large for scheduled window	Escalate to data engineer (SL 3)
Output dataset missing / stale	Operator impact: TM-10, Task 5-1	Check schedule; fix broken node; notify data steward
Schema mismatch after source change	Breaking change — operator impact	Escalate to SL 3 builder if multi-source; fix schema mapping

NOTE

When a pipeline fails, operators using TM-10, Task 5-1 see the failure in their data views and in Workshop applications. Fix pipeline issues promptly and document what failed and why. If the root cause is outside your SL 2 capability (e.g., requires @incremental logic, complex deduplication, Python transforms), escalate to a SL 3 builder or SL 4 developer.

1. After fixing the issue, click **Build Now** to confirm the fix resolves the failure.
2. Document the failure and fix in the pipeline's description field (right panel, **Edit Description**).
3. If the issue cannot be resolved at SL 2 level, escalate to your data engineer with the full error message, pipeline RID, and steps already attempted.

TASK 3-6: DOCUMENT A PIPELINE

TASK: Complete documentation for a pipeline before branch merge or handoff.

CONDITIONS: Pipeline is building successfully; builder is preparing for branch merge or handoff.

STANDARDS: The pipeline description will include source name, output dataset path, refresh schedule, data steward POC, creation date, and a plain-English description of what the pipeline does. Known data quality limitations are noted.

EQUIPMENT: MSS account with Editor role; pipeline open in Pipeline Builder.

DURATION: 15--20 minutes.

PROCEDURE:

1. Open the pipeline in Pipeline Builder.
2. In the right panel (no node selected), click **Edit Description**.
3. Enter the required elements:

```
SOURCE: [name and system of origin]
OUTPUT: [Compass path of output dataset]
SCHEDULE: [frequency and UTC time]
DATA STEWARD: [name and unit]
CREATED: [date] by [name, unit]
LAST MODIFIED: [date] by [name, unit]

PURPOSE: [one to two sentence plain-English description of what this
pipeline does and why it exists]

KNOWN LIMITATIONS: [any data quality issues, gaps, or edge cases]
```

1. Click **Save**.

NOTE

Documentation is not optional. Pipelines without descriptions will be flagged during governance reviews. Your replacement needs to understand what you built without having to contact you.

CHAPTER 4 — ONTOLOGY UI BASICS

4-1. WHAT THE ONTOLOGY IS

The Foundry Ontology is the semantic layer of MSS — it translates raw data tables into meaningful, interconnected objects. Instead of working with a table of rows and columns, users and applications work with Objects: a `UnitStatus` object, a `SoldierReadiness` object, a `MaintenanceRecord` object. Each Object Type has Properties (like fields), Link Types that connect it to other Object Types, and Actions that allow users to interact with it.

Build the Ontology last, after your data is clean. The Ontology reads from your curated dataset. If your curated dataset has schema issues, the Ontology configuration will fail or produce unreliable objects.

NOTE

Before creating any new Object Type, check the Ontology Manager to confirm no existing type already covers your use case. USAREUR-AF maintains established design patterns for common operational Object Types — consult your team lead or data steward before designing from scratch.

NOTE

The Object Types and properties you configure in the Ontology become what operators see in Workshop applications and in Quiver (TM-10, Task 5-3, Use Quiver to Explore Ontology Objects). A poorly designed Object Type — unclear property names, missing properties, wrong cardinality — creates friction for every operator who uses it. Refer to TM-10, Chapter 4 and Task 5-3 to understand the operator experience of your Ontology design before publishing.

4-2. ONTOLOGY MANAGER INTERFACE OVERVIEW

UI Area	Location	Purpose
Object Types panel	Left sidebar	List of all Object Types in the Ontology
Object Type editor	Center	Configure properties, primary key, backing dataset
Properties tab	Center tab	Add/edit/remove properties (columns)

UI Area	Location	Purpose
Links tab	Center tab	Configure Link Types to other Object Types
Actions tab	Center tab	Configure Actions (write-back operations)
Branch selector	Top toolbar	Current branch — always verify before editing
Publish button	Top toolbar	Merges your branch changes to the main Ontology

CAUTION

The Ontology is shared across all Workshop applications in your environment. A change to a shared Object Type affects every Workshop app and every user who reads that Object Type. Always work on a development branch. Never edit the main Ontology branch directly. See Chapter 7 for branching.

TASK 4-1: CREATE AN OBJECT TYPE

TASK: Create a new Object Type in the Ontology Manager backed by a curated dataset.

CONDITIONS: Builder has Editor access on the Ontology; a curated dataset backing the Object Type is published and has a stable schema; builder is working on a development branch.

STANDARDS: The builder will create an Object Type backed by the correct curated dataset, with a primary key configured, at least five meaningful properties added and named to standard, a plain-English description entered, and the Object Type previewing correctly (objects visible, properties show expected values) before the branch is submitted for merge.

EQUIPMENT: MSS account with Ontology Editor role; Ontology Manager access; curated dataset path; development branch active.

DURATION: 30--60 minutes.

PROCEDURE:

1. In the left navigation, open **Ontology Manager**.
2. Confirm the branch selector (top toolbar) shows your development branch — not `main`.
3. Click **New Object Type** (top right of the Object Types panel).
4. Enter the Object Type name in PascalCase singular noun format (e.g., `UnitStatus`, `MaintenanceRecord`).
5. Enter a description: include what this object represents, which data source backs it, the owning unit, and the data steward POC.
6. Click **Create**.

7. The Object Type editor opens. Click the **Properties** tab.
8. Click **Link to Dataset** (or **Backing Dataset**).
9. Browse Compass to your curated dataset and click **Select**.
10. The platform reads the dataset schema and lists available columns. Map columns to properties:
    - To add a property: click **Add Property**, select the column from the dataset, enter a display name (plain English, e.g., "Unit Name" not "unit_nm"), and select the data type (String, Number, Boolean, Date, etc.).
    - Repeat for each column you want to expose as a property.
11. Configure the **Primary Key**: click on the property that uniquely identifies each object (e.g., `unit_id`) and toggle **Primary Key** to ON.
12. Click **Save**.
13. Click **Preview** to confirm objects are visible and property values are correct.

> **NOTE**
>
> The primary key must be unique per object. If your backing dataset has duplicate values in the primary key column, objects will not display correctly. Fix the dataset before configuring the Ontology.

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

> **CAUTION**
>
> Do not expose columns containing PII, classified data, or data marked above the authorization level of intended users. Review the data with your Data Steward before configuring properties. Exclude sensitive columns by not adding them as properties.

TASK 4-2: ADD PROPERTIES TO AN OBJECT TYPE

TASK: Add, rename, or remove properties on an existing Object Type.

CONDITIONS: Object Type has been created; builder is on a development branch; additional properties need to be added or existing properties renamed.

STANDARDS: All added properties have plain-English display names, correct data types, and are formatted appropriately. Properties no longer needed are removed (not just hidden).

EQUIPMENT: MSS account with Ontology Editor role; Object Type open in Ontology Manager; development branch active.

DURATION: 20--30 minutes.

PROCEDURE:

Add a new property:

1. Open the Object Type in Ontology Manager (on your development branch).
2. Click the **Properties** tab.
3. Click **Add Property**.
4. Select the source column from the dataset column list.
5. Enter the display name (plain English, title case: "Report Date" not "rpt_dt").
6. Select the data type from the dropdown.
7. (Optional) Add a description for non-obvious fields.
8. Click **Save Property**.

Rename a property:

1. Click the property in the Properties tab.
2. Edit the **Display Name** field.
3. Click **Save**.

> **NOTE**
>
> Renaming a property changes what Workshop apps display — it does not change the backing column name. Applications referencing the old display name may need updating.

Remove a property:

1. Click the property in the Properties tab.
2. Click **Delete Property** (three-dot menu or trash icon).
3. Confirm the deletion.

> **CAUTION**
>
> Removing a property removes it from all Workshop apps that reference that Object Type. Check with your team lead and review all applications before removing any property from a shared Object Type.

NOTE

Interface design on Object Types is a SL 3 (Advanced Builder) task. At SL 2 level, you should understand that Interfaces exist — they define a shared contract of properties across multiple Object Types, enabling reusable Workshop components. For Interface creation procedures, see TM-30, Chapter 4.

TASK 4-3: CONFIGURE AN OBJECT VIEW

TASK: Configure an Object View for an Object Type to define how its properties are displayed to operators in Object Explorer and Workshop applications.

CONDITIONS: Object Type exists with at least five properties configured; builder is on a development branch; builder understands which properties are most operationally relevant to operators.

STANDARDS: Builder configures an Object View with a designated title property, a description property, and a prioritized list of displayed properties in a logical operational order. Object View is verified in Object Explorer before branch submission.

EQUIPMENT: MSS account with Ontology Editor role; Object Type open in Ontology Manager; development branch active.

DURATION: 20–30 minutes.

PROCEDURE:

What an Object View is:

An Object View defines the display configuration for an Object Type — which properties appear when operators view an object in Object Explorer, Workshop detail panels, or Quiver. By default, all properties are displayed in the order they were added. An Object View lets you control which properties are shown, in what order, and which property serves as the object's title (the primary human-readable identifier).

A well-configured Object View makes objects immediately recognizable to operators. A poorly configured Object View exposes internal field names, system codes, and low-priority properties that clutter the display and confuse users.

Step 1 — Open the Object View editor:

1. Open the Object Type in Ontology Manager (on your development branch).
2. Navigate to the **Object View** tab or **Views** section.
3. Click the default Object View to edit it, or click **New Object View** to create a named view for a specific audience.

Step 2 — Configure the title property:

1. Under **Title Property**, select the property that best identifies each object to an operator. This is the name that appears as the object's label in lists, tables, and pop-ups.
2. Good title properties: unit name, equipment bumper number, Soldier last name + last 4
3. Poor title properties: internal system IDs, numeric codes without context, UUIDs
4. Under **Description Property** (or **Subtitle**), select a secondary property that provides supporting context at a glance (e.g., unit name + status, or equipment type + location).

Step 3 — Configure displayed properties and order:

1. In the property list, check or uncheck properties to include or exclude them from the Object View.
2. Drag properties into the priority order that makes operational sense:
3. Most important / most frequently needed properties at the top
4. Administrative or system fields at the bottom or excluded entirely
5. Exclude properties that contain internal system codes, timestamps used only for pipeline logic, or fields with no operational meaning to the end user.

Step 4 — Verify the Object View:

1. Save the Object View.
2. Click **Preview** on the Object Type and select a sample object.
3. Verify the object displays with the correct title, description, and property order.
4. Mentally test: if an operator opened this object, would they immediately understand what they are looking at and find the information they need? If not, adjust the order and selection.

NOTE

Object Views affect what operators see in Object Explorer and in Workshop Object Explorer widgets. Workshop table widgets have their own column configuration — the Object View does not override Workshop widget display settings. However, the Object View does control the default property display for pop-ups and linked-object panels.

TASK 4-4: USE OBJECT EXPLORER TO VALIDATE OBJECT DATA

TASK: Use Object Explorer to inspect individual objects and their property values to verify that pipeline and Ontology configuration is producing correct, expected results before publishing a Workshop application.

CONDITIONS: Object Type has been created and is backed by a curated dataset; builder is on a development branch; pipeline has run and objects are being generated; builder needs to confirm object data is correct before proceeding to Workshop build.

STANDARDS: Builder opens Object Explorer for the target Object Type, searches for at least three specific objects, inspects their property values against known source data, confirms Link Type relationships are resolving correctly, and documents any discrepancies found before proceeding.

EQUIPMENT: MSS account with Ontology Editor role; Object Type published (on development branch); Object Explorer access.

DURATION: 30–45 minutes.

PROCEDURE:

What Object Explorer is:

Object Explorer is the Foundry interface for browsing and inspecting individual objects and their property values. It is a validation and debugging tool — not an end-user-facing product. Operators (SL 1) use Workshop applications to view data; builders use Object Explorer to confirm that data is correctly structured in the Ontology before publishing the application that exposes it.

Use Object Explorer when: - You want to verify a specific object's property values after a pipeline run - You want to confirm Link Types are resolving correctly (an object links to the expected related objects) - You are debugging why a Workshop widget is showing unexpected data - You are doing a spot-check after an Ontology change before promoting to production

Do NOT use Object Explorer as a substitute for Workshop. It is a builder validation tool. Do not direct operators to use Object Explorer to access data — that is what Workshop applications are for.

Step 1 — Open Object Explorer:

1. In the left navigation or platform search, find and open **Object Explorer**.
2. In the Object Type list (left panel), locate and select your Object Type.
3. Object Explorer loads a list of all objects of that type visible to you.

Step 2 — Search for specific objects:

1. Use the search bar at the top of Object Explorer to search for a specific object by the title property value (e.g., a bumper number, a unit name, a Soldier identifier).
2. Alternatively, use the filter panel to narrow the object list by a property value.
3. Select the object from the list to open its detail view.

Step 3 — Inspect property values:

1. In the object detail view, review each property value:
2. Is the value present (not null)?
3. Is the value correct? Compare against the source system or source dataset.
4. Is the data type correct (a date shows as a date, not a numeric timestamp)?
5. Are status values within the expected set (FMC, PMC, NMC — not "fmc," "n/a," or blank)?
6. Inspect the **Links** section of the object detail: verify that linked objects appear and are the correct related objects.

7. Repeat for at least two additional objects across different status values or unit affiliations.

Step 4 — Document findings and resolve issues:

1. If property values are incorrect, trace the issue:
   - Wrong value → likely a pipeline data transformation issue (fix in Pipeline Builder)
   - NULL values → likely a mapping issue in the Ontology property configuration, or a missing value in the source data
   - Missing links → likely a foreign key mismatch (verify the link key values match between Object Types)
2. Fix identified issues on the development branch, rebuild the pipeline, and re-verify in Object Explorer before proceeding.
3. Do not publish a Workshop application until Object Explorer validation confirms objects are correctly populated.

NOTE

Object Explorer shows data as of the most recent pipeline run. If you rebuilt the pipeline to fix an issue, wait for the build to complete before re-checking Object Explorer — the display reflects the last successful build output, not the in-progress build.

TASK 4-5: CREATE A LINK TYPE

TASK: Create a Link Type between two Object Types using the Ontology Manager UI.

CONDITIONS: Both Object Types to be linked exist and have primary keys configured; a foreign key relationship exists in the underlying data; builder is on a development branch.

STANDARDS: The builder will create a Link Type that correctly represents the relationship between the two Object Types, configured with the correct foreign key mapping, and verified by previewing linked objects before submitting for merge.

EQUIPMENT: MSS account with Ontology Editor role; both Object Types existing with primary keys; development branch active.

DURATION: 30--45 minutes.

PROCEDURE:

1. Open the source Object Type in Ontology Manager (on your development branch).
2. Click the **Links** tab.
3. Click **Add Link Type**.

4. Enter a link name in camelCase verb phrase format (e.g., `assignedTo`, `reportedBy`, `locatedAt`).
5. Enter a description of the relationship (e.g., "Connects a SoldierReadiness record to the UnitStatus of the Soldier's parent unit").
6. Under **Target Object Type**, select the Object Type this link points to.
7. Under **Foreign Key Configuration**:
8. Select the column in this Object Type's dataset that contains the key linking to the target.
9. Select the primary key property of the target Object Type.
10. Set the **Cardinality**:
11. **Many-to-One** — many source objects link to one target (most common; e.g., many soldiers in one unit)
12. **One-to-One** — each source links to exactly one target
13. **Many-to-Many** — requires a junction dataset; consult SL 3
14. Click **Save Link Type**.
15. Preview the source Object Type: click an individual object and verify linked objects appear in the Links section.

NOTE

If linked objects do not appear in preview, verify that foreign key values in the source dataset match primary key values in the target dataset. A mismatch in data types (integer vs. string) is a common cause of broken links.

NOTE

For complex relationship modeling, consult your team lead or data steward before building. Incorrect link configurations are difficult to fix after they are in production use.

NOTE

If an Ontology design requires any of the following, it exceeds SL 2 scope and must be escalated to a SL 3 advanced builder: (1) Many-to-many Link Types with complex junction logic; (2) Multi-step Actions with conditional routing or approval chains; (3) Derived properties requiring formula logic beyond the basic UI; (4) Ontology models that feed coalition-facing or MPE data products. Refer to TM-30, Chapter 4 (Ontology Design Methodology) for the SL 3 design process, and to TM-30, Chapter 4 (Ontology Design Through the UI), Task 4-3 for complex Action patterns.

4-5. ACTION TYPES OVERVIEW

Before creating an Action, understand which Action type matches your requirement. The platform supports multiple Action types. At SL 2 level, only write-back and form-based Actions are in scope. More complex types require SL 3 or SL 4 qualifications.

Write-back Actions update an existing Object's properties when executed. A user selects an object, triggers the Action, and a property value is updated in the backing dataset. This is the most common SL 2 Action type.

Form Actions present a user-facing form that accepts input and writes one or more values to an Object. The form may write to an existing object (Modify Object) or create a new one (Create Object). All SL 2 Actions are form-based at their core.

Webhook Actions trigger an external integration when executed — sending a notification, calling an external API, or initiating a downstream workflow. These require external endpoint configuration and are SL 4 scope (SL 4L, Software Engineer, or SL 4H, AI Engineer).

Conditional Actions execute different logic based on the current state of an object — for example, routing to different outcomes depending on an approval status field. Conditional routing is SL 3 scope (TM-30, Chapter 4, Task 4-3). Do not attempt to configure conditional routing at SL 2 level.

Batch Actions apply the same operation to multiple objects simultaneously. Batch Actions require careful validation and governance review because a single incorrect configuration can corrupt many records. Batch Actions are SL 3 scope.

Action Type	Description	In Scope
Write-back / Form (single-step)	Update a property or create an object via a form	SL 2
Form with validation rules	Required fields, allowed values, format checks	SL 2
Conditional routing	Different outcomes based on object state	SL 3
Multi-step (sequential submission)	Multiple form steps, approval chain	SL 3
Webhook / external integration	Triggers an external API or system	SL 4
Batch (multi-record write)	Applies operation to many objects at once	SL 3
TypeScript-driven (custom code)	Complex business logic in TypeScript	SL 4

If your requirement cannot be met with a single-step form-based write-back Action, stop and escalate to a SL 3 builder. Do not attempt to approximate complex Action logic with multiple simple Actions — this creates data integrity risks and governance problems.

TASK 4-6: CREATE AN ACTION

TASK: Create a single-step, form-based Action on an Object Type using the Ontology Manager UI.

CONDITIONS: The target Object Type exists and is correctly configured; builder is on a development branch; the write-back dataset is prepared; Data Steward has approved write-back for this Object Type.

STANDARDS: The builder will create a form-based Action that allows authorized users to write a value back to the backing dataset, with required fields marked, input validation configured, and the Action tested end-to-end before requesting a merge.

EQUIPMENT: MSS account with Ontology Editor role; Object Type with backing dataset; write-back authorization from Data Steward; development branch active.

DURATION: 30--60 minutes.

PROCEDURE:

1. Open the target Object Type in Ontology Manager (on your development branch).
2. Click the **Actions** tab.
3. Click **New Action**.
4. Enter the Action name as a plain English verb phrase (e.g., "Submit SITREP Update", "Mark Maintenance Complete").
5. Enter a description: what does this Action do, who should use it, what does it write.
6. Under **Action Type**, select **Modify Object** (updating existing objects) or **Create Object** (creating new ones).
7. Under **Form Configuration**:
8. Click **Add Form Field** for each input the user needs to provide.
9. For each field: select the property it writes to, enter the field label, select the input type (text, dropdown, date picker, checkbox), and toggle **Required** if the field must not be blank.
10. For dropdown fields: enter the list of allowed values.
11. Under **Write Target**: confirm the backing dataset this Action writes to.
12. Under **Permissions**: select which Groups or Roles are allowed to execute this Action. Do not leave Actions open to all users unless explicitly authorized.
13. Click **Save Action**.
14. Test the Action: in the preview, find an object, click the Action name, complete the form, submit, then check the backing dataset to confirm the update was written.

CAUTION

Actions that write to datasets affect all downstream applications. Before enabling a write-back Action in production, test it on a development branch with test data — not with live operational records.

NOTE

Multi-step Actions with conditional routing, sequential submission steps, approval chains, or multi-record writes are SL 3 scope — refer to TM-30, Chapter 4, Task 4-3. These do NOT require code (TypeScript is SL 4 scope). SL 2 Actions are single-step: operator fills a form, one field in the backing dataset is updated.

CHAPTER 5 — BUILDING WORKSHOP APPLICATIONS

5-1. WHAT WORKSHOP IS

Workshop is MSS's drag-and-drop application builder. Using Workshop, you assemble widgets — tables, charts, filters, forms, maps — into applications that end users interact with daily. No code is required.

Workshop applications read from the Ontology. They do not read directly from datasets. This means your Ontology must be correctly configured (Chapter 4) before you begin building in Workshop.

The build order:

```
Data (Pipeline Builder) -> Ontology (Object Types, Links, Actions) -> Workshop App
```

NOTE

The Workshop applications you build are consumed by operators working from SL 1. Before building, read TM-10, Chapter 4 (Using Workshop Applications) — specifically Task 4-1 (Orient to a Command-Level Application), Task 4-3 (Apply Filters to a Dashboard), Task 4-4 (Submit Data Using an Action Form), and Task 4-5 (Execute an Action Button). Build your application so an operator following those SL 1 tasks can use it without confusion.

5-2. WORKSHOP WIDGET LIBRARY

Widget	Use For	Key Configuration
Table	Display rows of Object data; sort, filter, select	Object Type source; columns to display; sort order
Chart — Bar	Compare values across categories	X-axis (category property), Y-axis (numeric property)
Chart — Line	Show trends over time	X-axis (date property), Y-axis (numeric property)
Chart — Pie/Donut	Show proportion/composition	Segment property, value property
Metric Tile	Display a single KPI number	Aggregation (count, sum, avg) on a property

Widget	Use For	Key Configuration
Filter — Dropdown	Let users filter displayed data	Property to filter on; allowed values or dynamic
Filter — Date Range	Filter by date window	Date property; default range
Filter — Search	Free-text search across an Object Type	Property to search; Object Type
Form	Allow users to submit Actions	Action to invoke; fields from Action configuration
Map	Display objects at geographic coordinates	Latitude/longitude or address property
Object Explorer	Show properties of one selected object	Object Type; properties to display
Rich Text	Add labels, instructions, or context	Static or template-driven text
Container	Group and layout other widgets	Layout (horizontal, vertical, grid)
Conditional Visibility	Show/hide widgets based on filter state	Condition expression

5-3. WORKSHOP INTERFACE OVERVIEW

UI Area	Location	Purpose
Canvas	Center	Where you place and arrange widgets
Widget Library	Left panel	Available widgets — drag to canvas
Widget Properties	Right panel	Configure the selected widget
Variables panel	Left panel (tab)	Define app-level variables and filters
Preview mode	Top toolbar toggle	Switch between edit and user view
Publish button	Top toolbar	Make the app available to users
Branch selector	Top toolbar	Confirm you are on a development branch

NOTE

When designing filter panels and navigation, test them from the operator's perspective. An operator following TM-10, Task 4-3 (Apply Filters to a Dashboard) expects predictable filter behavior and clear labeling. Test your application against TM-10 Chapter 4 tasks before publishing.

TASK 5-1: CREATE A WORKSHOP APPLICATION

TASK: Create a new Workshop application in the correct project folder.

CONDITIONS: Builder has Workshop Builder permission; at least one Object Type exists and is correctly configured; builder is on a development branch; the application's purpose and intended users are defined.

STANDARDS: The builder will create a new Workshop application in the correct project folder, named to standard, connected to the appropriate Object Type, with a basic layout (at minimum a title and one data widget) that renders correctly in Preview mode.

EQUIPMENT: MSS account with Workshop Builder permission; Object Type configured and previewing with data; development branch active.

DURATION: 45--90 minutes.

PROCEDURE:

1. In Compass, navigate to your project's `applications/` folder.
2. Click **New**, then **Workshop Application**.
3. Enter the application name in plain English, unit-appropriate format (e.g., `EUCOM SITREP Dashboard`).
4. Click **Create**. Workshop editor opens.
5. Confirm the branch selector shows your development branch.
6. From the Widget Library (left panel), drag a **Container** widget onto the canvas for your header.
7. Inside the container, drag a **Rich Text** widget. Enter the application title and context text.
8. Drag a **Table** widget below the header container.
9. In the right panel under **Data Source**: click **Select Object Type**, browse to your Object Type, and confirm.
10. Under **Columns**: select the properties to display. Add only the columns end users need.
11. Click **Preview** (top toolbar) to switch to user view and confirm the table renders with data.
12. Switch back to **Edit** mode to continue building.

NOTE

Workshop apps built on development branches are only visible to team members with project access. They are not visible to end users until published (Task 5-7).

TASK 5-2: ADD FILTER WIDGETS AND CONNECT TO DATA

TASK: Add filter widgets to a Workshop application and connect them to data widgets.

CONDITIONS: A Workshop application exists with at least one data widget; the Object Type has filterable properties; builder is in Workshop edit mode on a development branch.

STANDARDS: The builder will add filters allowing end users to narrow data by at least two dimensions (e.g., unit and date range), and confirm that selecting a filter value updates all connected widgets in Preview mode.

EQUIPMENT: MSS account with Workshop Builder permission; Workshop application open in edit mode; development branch active.

DURATION: 30--45 minutes.

PROCEDURE:

Add a Dropdown Filter:

1. From the Widget Library, drag **Filter — Dropdown** onto the canvas.
2. In the right panel under **Object Type**: select the same Object Type as your data widget.
3. Under **Filter Property**: select the property to filter on (e.g., `unit_name`, `status`, `country_code`).
4. Under **Display Label**: enter a plain-English label (e.g., "Filter by Unit").
5. Under **Default Value**: set a default if appropriate, or leave blank for "Show All."
6. Connect the filter to the data widget: click the data widget, in the right panel click **Filters**, then **Add Filter**, and select the dropdown filter variable.
7. Preview: select a value in the dropdown filter and verify the table updates.

Add a Date Range Filter:

1. Drag **Filter — Date Range** onto the canvas.
2. Select the Object Type and the date property to filter on (e.g., `report_date`).
3. Set a default range appropriate for the data's refresh cadence.
4. Connect to the data widget (same as above).
5. Preview: adjust the date range and verify the table updates.

NOTE

Filters work across all widgets connected to the same Object Type variable. If you want independent filtering per widget, create separate Object Type variables for each widget.

TASK 5-3: ADD A TABLE WIDGET

TASK: Add and configure a Table widget to display Object Type data in a Workshop application.

CONDITIONS: A Workshop application exists; the Object Type is correctly configured with properties; builder is in edit mode on a development branch.

STANDARDS: The builder will add a Table widget connected to the correct Object Type, displaying only the columns end users need, with correct column labels, and rendering with accurate data in Preview mode.

EQUIPMENT: MSS account with Workshop Builder permission; Workshop application open in edit mode; Object Type with properties configured; development branch active.

DURATION: 20--30 minutes.

PROCEDURE:

1. From the Widget Library, drag **Table** onto the canvas.
2. In the right panel under **Data Source**: click **Select Object Type** and browse to the target Object Type.
3. Under **Columns**: click **Add Column** and select properties to display.
4. For each column, review the column label — update to plain English if the property display name is not user-friendly.
5. Under **Default Sort**: select a column and sort direction appropriate to the data (e.g., sort by report date descending for time-series data).
6. Connect any applicable filters (see Task 5-2).
7. Click **Preview** and verify the table renders with data and column labels are correct.

NOTE

Display only columns the end user needs. A table with 20 columns showing raw field names is harder to read than a 6-column table with plain-English labels. Remove unnecessary columns.

TASK 5-4: ADD A CHART WIDGET

TASK: Add and configure chart widgets to visualize Object Type data in a Workshop application.

CONDITIONS: A Workshop application exists; Object Type has appropriate numeric and categorical or date properties; builder is in edit mode on a development branch.

STANDARDS: The builder will add at least one chart that meaningfully visualizes a metric, correctly configured with labeled axes, and rendering with accurate data in Preview mode.

EQUIPMENT: MSS account with Workshop Builder permission; Workshop application open in edit mode; Object Type with numeric and categorical properties; development branch active.

DURATION: 30--45 minutes.

PROCEDURE:

Add a Bar Chart:

1. From the Widget Library, drag **Chart — Bar** onto the canvas.
2. In the right panel under **Data Source**: select the Object Type.
3. Under **X-Axis**: select a categorical property (e.g., `unit_name`, `country`, `status_category`).
4. Under **Y-Axis**: select a numeric property or aggregation (e.g., Count of objects, Sum of `personnel_strength`).
5. Under **Color By** (optional): select a property to color-code bars (e.g., `readiness_status`).
6. Enter a **Chart Title** (e.g., "Readiness by Unit").
7. Connect any applicable filters.
8. Preview to confirm bars display with correct data.

Add a Line Chart (trend over time):

1. Drag **Chart — Line** onto the canvas.
2. Data Source: your Object Type.
3. X-Axis: your date property. Enable **Date Grouping** if available.
4. Y-Axis: your numeric property or count.
5. Title: plain English (e.g., "Daily SITREP Submissions — Last 30 Days").
6. Connect filters. Preview.

Add a Metric Tile (single KPI number):

1. Drag **Metric Tile** onto the canvas.
2. Data Source: Object Type.
3. Metric: select aggregation — Count, Sum, or Average of a numeric property.
4. Label: plain English (e.g., "Total Units Reporting", "Average Readiness %").

5. (Optional) Configure conditional color: green/yellow/red thresholds.
6. Preview to confirm the value is correct.

TASK 5-5: CONFIGURE AN ACTION FORM WIDGET

TASK: Add a Form widget connected to an Action in a Workshop application.

CONDITIONS: An Action is configured on the relevant Object Type (Task 4-6); builder is in Workshop edit mode on a development branch; intended users and permissions are confirmed.

STANDARDS: The builder will add a Form widget connected to the correct Action, with all required fields visible and labeled, and confirm that submitting a test entry through the form writes data to the backing dataset.

EQUIPMENT: MSS account with Workshop Builder permission; Action configured on the Object Type; Workshop application open in edit mode; development branch active.

DURATION: 30--45 minutes.

PROCEDURE:

1. From the Widget Library, drag **Form** onto the canvas.
2. In the right panel under **Action**: click **Select Action**, browse to the Object Type, and select the Action (e.g., "Submit SITREP Update").
3. The form fields defined in the Action configuration will appear automatically.
4. Review the field labels. If any are not user-friendly, update them in the Action configuration in Ontology Manager — changes propagate automatically.
5. (Optional) Add a **Rich Text** widget above the form with instructions for the user.
6. Preview the application. Complete the form with test data and submit.
7. Open the backing dataset in Compass, click **Preview**, and confirm the test row was written correctly.
8. If the row did not appear: check the Action configuration for write target issues. Do not publish until write-back is confirmed working.

CAUTION

Test Action forms with non-operational test data. Do not submit live operational data through an unvalidated form on a development branch.

TASK 5-6: CONFIGURE LAYOUT AND ORGANIZE A WORKSHOP APPLICATION

TASK: Organize the layout of a Workshop application for usability.

CONDITIONS: Core widgets are added; builder is in edit mode; layout needs to be organized for usability.

STANDARDS: The application will have a clear visual hierarchy (title, filters, data displays, forms), widgets sized appropriately and aligned, and the layout usable on standard government workstation screen sizes (1920x1080).

EQUIPMENT: MSS account with Workshop Builder permission; Workshop application open in edit mode; core widgets placed.

DURATION: 30--45 minutes.

PROCEDURE:

1. In Workshop settings (gear icon), confirm the layout is set to **Fixed Width** (1200px or 1440px) for desktop use.
2. Use Container widgets for structure: group all filters in one horizontal container at the top; charts in a grid below; table beneath charts.
3. Resize widgets by clicking and dragging widget edges. Standard layout:
4. Header container: full width, 80px height
5. Filter row: full width, 60px height
6. Metric tiles row: full width, divided into 3-4 equal tiles
7. Charts: 50% width side-by-side, or full width for single chart
8. Table: full width, 400-600px height
9. Label all widgets: every chart and metric tile should have a title. Every filter should have a label.
10. Add a Rich Text header: include application name, classification/handling markings, owning unit, data steward POC, and "Data as of [date]."
11. Toggle Preview mode and review at 100% zoom. Check for overlapping widgets, cut-off text, or widgets too small to read.
12. Fix any layout issues before publishing.

NOTE

Test the application at 1920x1080 resolution — the standard government workstation. Also check whether the layout is functional on tablet-size screens used in some TOC environments.

TASK 5-7: PUBLISH A WORKSHOP APPLICATION

CAUTION

Before publishing, assess whether your application design is within SL 2 scope. If your design includes: multiple pages with conditional navigation between them; widgets that pass parameters to other widgets; role-based conditional layouts — your application is likely SL 3 scope. Refer to TM-30, Chapter 2 (Advanced Workshop Application Design), specifically Section 2-1 (The Multi-Page Application Model), to determine whether your design should be escalated to a SL 3 qualified builder before publication.

TASK: Publish a completed Workshop application and configure access for intended users.

CONDITIONS: Workshop application is complete; all widgets render correctly in Preview; branch has been merged (Chapter 7); Data Steward has reviewed and approved publication; access list is defined.

STANDARDS: The builder will publish the application, configure access so only authorized users can open it, verify that a test user (View-only access) can access and use the application, and notify intended users.

EQUIPMENT: MSS account with Workshop Builder permission; completed Workshop application on main branch; approved access list from Data Steward.

DURATION: 30--45 minutes.

PROCEDURE:

1. Confirm the branch has been merged to main (Chapter 7). You cannot publish from a development branch.
2. Open the Workshop application in Compass (on main branch).
3. Click **Publish** (top toolbar).
4. Review the publish summary: confirm the application name, version number, and Object Types it reads from.
5. Click **Confirm Publish**.
6. After publishing, click **Share** (or **Access Settings**).
7. Under **Who can access this application**:
8. Click **Add**, search for the Group or individual user.
9. Assign **View** access for end users.
10. Assign **Edit** access only to other builders who need to modify the application.
11. Set **Link Sharing** to **Off** unless C2DAO has specifically authorized link-based access.
12. Click **Save**.
13. Test access: ask a colleague with View-only access to confirm they can open and use the application.

4. Notify intended users via official channels. Include what the app shows and who to contact for issues.

CAUTION

Do not publish applications containing data with markings above the access level of intended users. Confirm classification, handling, and need-to-know before publishing to any group.

NOTE

Compass links to Workshop applications are permanent once published. Do not delete or move a published application without notifying users and coordinating with your Data Steward.

CHAPTER 6 — ANALYSIS WITH CONTOUR AND QUIVER

6-1. CONTOUR VS. QUIVER — WHEN TO USE EACH

Tool	Purpose	Best For	SL 2 Scope
Contour	Interactive analysis — build and save views of data	Analysts exploring a dataset; saved analyses shared with a small audience	Build saved analyses using point-and-click interface
Quiver	Dashboard builder — assemble charts and metrics into a shareable page	Quick executive dashboards; simple multi-chart views	Build basic dashboards with charts and metric tiles

Use Contour when you need to explore a dataset, build a table/chart for analysis, and save that view for reuse.

Use Quiver when you need a simple, shareable dashboard with a few metrics and charts and do not need the full Workshop widget library or Action capability.

Use Workshop when you need a full interactive application with filters, forms, Actions, and complex layout.

NOTE

Operators interact with Contour and Quiver using TM-10, Task 5-2 (Use Contour for No-Code Analysis) and Task 5-3 (Use Quiver to Explore Ontology Objects). When building saved analyses or Quiver configurations, understand the operator's analysis workflow from SL 1. Build analyses that support workflows operators actually perform. For advanced Contour capabilities (formula editor, multi-table aggregations, pivot analysis), refer to TM-30, Chapter 5 (Advanced Analytics: Contour and Quiver).

6-2. BUILDING SAVED ANALYSES IN CONTOUR

TASK 6-1: BUILD A SAVED ANALYSIS IN CONTOUR

TASK: Create a saved Contour analysis answering a specific operational question.

CONDITIONS: Builder has access to the target dataset or Object Type; the analysis purpose and intended audience are defined.

STANDARDS: The builder will create a saved Contour analysis answering a specific operational question, containing at least one table view and one chart, saved with a descriptive name, and shared with the intended audience.

EQUIPMENT: MSS account with Contour access; source dataset or Object Type available.

DURATION: 30--60 minutes.

PROCEDURE:

1. In the MSS left navigation, open **Contour**.
2. Click **New Analysis**.
3. Name the analysis (plain English, descriptive: e.g., `VCORPS SITREP Submission Rate — Weekly`).
4. Under **Data Source**, click **Add Dataset or Object Type**. Search for and select your target.
5. Contour opens the data view with all columns visible.

Build a Table View:

1. The default view is a table showing all rows and columns.
2. To filter rows: click the column header, select **Filter**, then condition and value.
3. To select only needed columns: click **Columns** (top toolbar) and uncheck columns to hide.
4. To sort: click a column header, then **Sort Ascending/Descending**.
5. To group and aggregate: click **Group By**, select the grouping column, then click **Aggregate** and add an aggregation (Count, Sum, Average) on a numeric column.

Build a Chart View:

1. Click the **+** tab (next to the current view tab) to add a new view.
2. Select **Chart**.
3. Under **Chart Type**: select Bar, Line, or Pie.
4. Configure X-axis and Y-axis (same logic as Workshop charts — see Task 5-4).
5. Add a title to the chart tab (double-click the tab name).

Save and Share:

1. Click **Save Analysis** (top toolbar).
2. Click **Share**, add authorized users or groups, and assign View access.
3. Copy the Contour analysis link and share with the intended audience via official channels.

NOTE

Contour analyses reflect current data when opened — they are not live-updating dashboards. For a continuously updated operational view, use Workshop.

TASK 6-2: ADD A PIVOT TABLE IN CONTOUR

TASK: Add a pivot table view to an existing Contour analysis.

CONDITIONS: A Contour analysis exists; builder needs to summarize data across two dimensions (e.g., unit by month, status by country).

STANDARDS: The builder will configure a pivot table that correctly summarizes data across the intended dimensions and verify totals against the base table before saving.

EQUIPMENT: MSS account with Contour access; existing Contour analysis open.

DURATION: 20--30 minutes.

PROCEDURE:

1. In the Contour analysis, click **+** to add a new view.
2. Select **Pivot Table**.
3. Under **Rows**: select the property to use as row labels (e.g., `unit_name`).
4. Under **Columns**: select the property to use as column headers (e.g., `report_month`, `country`).
5. Under **Values**: select the numeric property and aggregation function (Count, Sum, Average).
6. Review the pivot table. Verify row totals match the base table count for each unit.
7. Save the analysis.

6-3. BUILDING BASIC QUIVER DASHBOARDS

TASK 6-3: BUILD A QUIVER DASHBOARD

TASK: Create a Quiver dashboard with charts and metric tiles.

CONDITIONS: Builder has access to the target datasets or Object Types; intended dashboard audience is defined.

STANDARDS: The builder will create a Quiver dashboard with at least two charts and one metric tile, named correctly, and shared with the appropriate audience.

EQUIPMENT: MSS account with Quiver access; source datasets or Object Types available.

DURATION: 30--60 minutes.

PROCEDURE:

1. In the MSS left navigation, open **Quiver**.
2. Click **New Dashboard**.
3. Name the dashboard (e.g., `USAREUR-AF Readiness Summary — Weekly`).
4. Click **Add Widget**, then select **Chart**.
5. Configure the chart:
6. Under **Data Source**: select the dataset or Object Type.
7. Select Chart Type (Bar, Line, Pie).
8. Configure X-axis and Y-axis properties.
9. Enter a chart title.
10. Click **Add Widget** again, then select **Metric**.
11. Configure the metric tile:
12. Data source: Object Type.
13. Aggregation: Count, Sum, or Average.
14. Label: plain English name.
15. (Optional) Thresholds: set green/yellow/red values.
16. Arrange widgets by dragging. Resize by dragging widget edges.
17. Click **Save**.
18. Click **Share**, add authorized users, and assign View access.
19. Confirm a test user can open and view the dashboard.

> **NOTE**
>
> Quiver dashboards do not support Actions, complex filters, or form submission. Use Quiver for quick summary views; use Workshop for interactive operational applications.

TASK 6-4: CONFIGURE AUTOMATIC REFRESH ON A QUIVER DASHBOARD

TASK: Configure automatic refresh on a Quiver dashboard.

CONDITIONS: Quiver dashboard exists; underlying data updates on a schedule; users need to see current data without manually refreshing.

STANDARDS: The builder will configure automatic refresh at an interval appropriate to the data's update cadence, and confirm the last-refreshed timestamp is visible to users.

EQUIPMENT: MSS account with Quiver access; Quiver dashboard open in edit mode.

DURATION: 15--20 minutes.

PROCEDURE:

1. Open the Quiver dashboard in edit mode.
2. Click **Dashboard Settings** (gear icon, top right).
3. Under **Auto Refresh**: toggle ON.
4. Set the refresh interval to match or slightly exceed the pipeline schedule. Do not set refresh more frequently than the data actually updates.
5. Under **Show Last Refreshed Timestamp**: toggle ON.
6. Click **Save Settings**.

CHAPTER 7 — BRANCHING AND ENVIRONMENT MANAGEMENT VIA UI

7-1. WHY BRANCHING MATTERS

Branching is how MSS protects production data and applications from work-in-progress changes. Every resource in Foundry — datasets, Ontology configurations, Workshop apps — lives on a branch. The `main` branch is what users see and depend on. Your development branch is where you build and test.

The rule: never edit main directly. All SL 2 build work happens on a named development branch. When the work is tested and approved, you request a merge into main. A reviewer approves the merge and the changes go live.

Working without a branch is the equivalent of making changes to a live operational system without testing.

> **NOTE**
>
> Operators (SL 1) work only with the main/production branch of MSS resources. They do not see development branches. When you merge your development branch to main, operators immediately see the changes in their next refresh. A faulty merge directly affects operational users. Refer to TM-10, Chapter 7 (Troubleshooting and Support) to understand what operators experience when a bad merge breaks an application. Treat every merge to main as a production release.

7-2. FOUNDRY BRANCHING CONCEPTS

Term	Meaning
Branch	An isolated copy of the Ontology where you can make changes without affecting main
Main branch	The production branch — what all users see; never edit directly
Development branch	Your working branch — named `dev-[feature]` or `dev-[lastname]`
Check in	Save your changes to the branch (versioned save with a message)
Merge request	A formal request to integrate your branch changes into main

Term	Meaning
Review	A team lead or designated reviewer approves the merge request
Conflict	Two branches have made changes to the same resource — must be resolved before merging

TASK 7-1: CREATE A DEVELOPMENT BRANCH

TASK: Create a named development branch before making any Ontology changes.

CONDITIONS: Builder has Editor access on the Ontology; a build task requires changes to Object Types, Link Types, or Actions.

STANDARDS: The builder will create a named development branch before making any Ontology changes, confirm the branch is active (shown in the branch selector), and not make any changes on the main branch.

EQUIPMENT: MSS account with Ontology Editor role; Ontology Manager access.

DURATION: 10--15 minutes.

PROCEDURE:

1. Open **Ontology Manager**.
2. In the top toolbar, click the **Branch Selector** (shows the current branch name).
3. Click **New Branch**.
4. Enter the branch name:
5. Feature-based: `dev-sitrep-objecttype`
6. Person-based: `dev-smith` (use only for short-lived personal branches)
7. Under **Branch From**: select `main` (or the base branch your team lead specifies).
8. Click **Create Branch**.
9. Confirm the branch selector now shows your new branch name.
10. Begin Ontology edits only after confirming the correct branch is active.

TASK 7-2: TEST CHANGES IN A DEVELOPMENT BRANCH

TASK: Check in and verify changes on a development branch before submitting a merge request.

CONDITIONS: Builder has made changes to Object Types, Link Types, or Actions on a development branch and needs to save and verify progress.

STANDARDS: The builder will check in changes with a descriptive message at logical intervals, verify Object Type previews show correct data, and confirm no downstream applications are broken by the changes before requesting a merge.

EQUIPMENT: MSS account with Ontology Editor role; development branch with changes; Object Type preview access.

DURATION: 20--45 minutes.

PROCEDURE:

1. In Ontology Manager, with your development branch active, click **Check In** (top toolbar).
2. Enter a check-in message describing what you changed:
3. Good: "Add UnitStatus Object Type backed by sitrep_feed_curated; add unit_id primary key and 6 properties"
4. Poor: "changes" or "wip"
5. Click **Confirm Check In**.
6. Verify the changes are saved (the Check In button returns to inactive state).
7. Click **Preview** on each modified or created Object Type to confirm:
8. Objects are visible and count is correct.
9. Property values display correctly.
10. Link Types show linked objects as expected.
11. If a Workshop application uses the modified Object Type, open it in Preview mode and verify it still renders correctly.
12. Document any issues found and fix before requesting the merge.

NOTE

Check in frequently — at minimum after completing each Object Type, Link Type, or Action configuration. Small, frequent check-ins make it easier to identify and revert specific changes if something breaks.

TASK 7-3: SUBMIT A MERGE REQUEST FOR PRODUCTION PROMOTION

TASK: Submit a formal merge request to integrate development branch changes into main.

CONDITIONS: Build work on the development branch is complete; all changes have been checked in; builder has tested the changes; team lead is ready to review.

STANDARDS: The builder will submit a merge request with a complete description of all changes, dependencies or breaking changes noted, test results included, and the request directed to the correct reviewer. The builder will not approve their own merge request.

EQUIPMENT: MSS account with Ontology Editor role; completed, tested development branch; team lead identified as reviewer.

DURATION: 20--30 minutes.

PROCEDURE:

1. In Ontology Manager, with your development branch active, click **Merge Request** (top toolbar).
2. Under **Merge Into**: confirm the target branch is `main`.
3. Under **Description**, fill in:

```
CHANGES: [list each Object Type, Link Type, Action, or pipeline modified/created]
REASON: [operational need driving this change]
TEST RESULTS: [what you tested and results]
DEPENDENCIES: [any datasets, pipelines, or resources this change depends on]
BREAKING CHANGES: [yes/no; if yes, describe impact on existing applications]
```

1. Under **Reviewer**: select your team lead or designated Ontology reviewer. Do not assign to yourself.
2. Click **Submit**.
3. Notify your reviewer via official channels that a merge request is pending.

> **CAUTION**
>
> If your changes include modifications to an existing shared Object Type, call out any breaking changes explicitly in the merge request description. The reviewer needs to know whether downstream Workshop applications or analyses may be affected before approving.

TASK 7-4. RESOLVE A BRANCH CONFLICT VIA UI

CONDITIONS: A merge request has been flagged with a conflict; another branch modified the same resource before this merge was approved.

STANDARDS: The builder will identify the conflicting resource, understand what change the conflicting branch made, and resolve the conflict by selecting the correct version without discarding authorized changes made by another team member.

EQUIPMENT: MSS account with Ontology Editor role; merge request with flagged conflict.

DURATION: 20--45 minutes.

PROCEDURE:

1. Open the merge request in Ontology Manager.

2. Click **View Conflicts**. The UI shows which resources have conflicts.
3. For each conflicting resource, the UI displays two versions: **Your Changes** (your branch) and **Incoming Changes** (the other branch).
4. Review both versions. Understand what each change does.
5. Resolve the conflict:
6. If your change is correct: select **Keep My Version**.
7. If the incoming change is correct: select **Keep Incoming Version** and plan to redo your work.
8. If both changes are needed: contact your team lead — multi-field conflicts may require a data engineer (SL 3).
9. After all conflicts are resolved, click **Mark as Resolved**.
10. Resubmit the merge request.

NOTE

When in doubt on a conflict, do not guess. Contact the team member whose branch created the conflict. Resolving a conflict incorrectly can delete another team member's work or produce an inconsistent Ontology state.

CHAPTER 8 — BUILDER STANDARDS AND GOVERNANCE

8-1. OVERVIEW

NOTE

Builder standards exist because builders have elevated privileges that operators (SL 1) do not have. Before building, understand the security markings and access controls that govern operator data access (TM-10, Chapter 6, Security, Classification, and Markings). Your applications, pipelines, and Ontology configurations must respect those controls. For SL 3-level governance responsibilities on shared infrastructure, refer to TM-30, Chapter 7 (Data Governance and Lineage).

Builder standards are not optional. They exist to maintain data quality, operational reliability, and security across the USAREUR-AF MSS environment. All SL 2 builders are accountable for the quality and compliance of everything they publish.

8-2. NAMING CONVENTIONS (CONSOLIDATED)

See Chapter 2-3 for the full naming conventions table. Critical rules:

1. Dataset paths are permanent — choose carefully before creating.
2. Object Type names are PascalCase singular nouns — always.
3. Link Type names are camelCase verb phrases — always.
4. Pipeline names use kebab-case — always.
5. Workshop app names are plain English, unit-appropriate — always.
6. Do not use PII, classified terms, or operational codenames in any resource name or description.

8-3. C2DAO APPROVAL TRIGGERS

The following actions require explicit C2DAO approval before proceeding. Formal approval must be documented. Do not proceed based on informal authorization.

Action	Approval Required From	Documentation Required
Ingest a new external data source	C2DAO + Unit Data Steward	Data source registration form; data handling determination
Create a new top-level Project	C2DAO	Project authorization memo
Modify a shared Object Type used by multiple units	C2DAO + affected unit Data Stewards	Change notification; impact assessment
Publish a Workshop application to users outside your unit	C2DAO	Application publication request
Set Project Visibility to Organization-wide	C2DAO	Access policy waiver
Enable an Action (write-back) on a shared Object Type	C2DAO + Unit Data Steward	Write-back authorization
Schedule a pipeline pulling from a classified or sensitive source	C2DAO + security officer	Data handling certification

CAUTION

Proceeding without required approvals is a governance violation. It may result in revocation of builder access, a data incident report, and notification to your chain of command. When in doubt, ask your Data Steward before acting.

8-4. DATA QUALITY CHECKLIST

Before publishing any pipeline, Ontology configuration, or Workshop application, verify:

Pipeline / Dataset:

- ☐ Output dataset has a description (source, owner, refresh schedule, steward POC)
- ☐ Output row count is within expected range (checked against source)
- ☐ No unexpected null values in primary key column

- ☐ No unexpected duplicate rows
- ☐ Date columns contain valid date values (not string representations)
- ☐ Column names follow naming conventions (snake_case, no spaces, no special characters)
- ☐ Data contains no PII in columns not authorized for PII

Ontology:

- ☐ Object Type has a description
- ☐ Primary key is correctly configured (unique, non-null)
- ☐ All property display names are plain English
- ☐ Properties containing sensitive data are excluded or restricted appropriately
- ☐ Link Types are tested (linked objects appear in preview)
- ☐ Actions tested with test data (write-back confirmed working)

Workshop Application:

- ☐ Application has a title and data classification/handling marking
- ☐ All charts have titles
- ☐ All filters have labels
- ☐ Default filter state shows a useful, non-empty data view
- ☐ Form submissions tested with test data
- ☐ Application tested in Preview mode at 1920x1080
- ☐ Access permissions configured correctly
- ☐ Application tested by a user with end-user access (not builder access)

8-5. BUILDER ACCOUNTABILITY

As a SL 2 builder, you are personally accountable for:

1. **What you publish.** If your application shows incorrect data to a commander, that is a data quality failure. Test thoroughly.
2. **Who you give access to.** If you assign permissions that allow unauthorized access to operational data, that is a security failure. Follow least-privilege principles.
3. **What you ingest.** If you connect a data source without C2DAO approval, that is a governance violation. Get approval first.
4. **How you document.** If you leave undocumented pipelines and Object Types that break when you PCS, that is a maintainability failure. Document everything.

The four rules:

1. Always work on a branch — never on main/production directly.
2. Test before you publish — verify data, spot-check values, test with end-user credentials.
3. Follow naming conventions — every resource must be named to standard.
4. Document your work — every resource requires a description.

NOTE

All builder actions in MSS — pipeline creation and modification, Ontology changes, Workshop application publishing, branch creation, and merge requests — are logged with your credentials, timestamp, and the specific change made. These logs are retained for accountability reviews, security audits, and incident investigation. You are personally accountable for all changes made under your credentials.

8-6. CUI HANDLING

NOTE

CUI (Controlled Unclassified Information) — If a data source you are ingesting contains CUI (e.g., personnel records, financial data, acquisition information), do NOT ingest it without first coordinating with your unit data steward and confirming: (a) The data is authorized for MSS ingestion at the appropriate classification level. (b) The output dataset will be marked and access-controlled appropriately. (c) Any downstream Workshop applications exposing CUI are restricted to authorized personnel only. When in doubt, treat the data as CUI. Contact your data steward before proceeding.

CHAPTER 9 — TROUBLESHOOTING AND COMMON ERRORS

9-1. OVERVIEW

BLUF: When a build fails or produces unexpected output, diagnose methodically before escalating. Most issues fall into four categories: pipeline errors, Ontology configuration errors, Workshop display errors, and access/permissions errors.

Do not guess at the cause. Follow the diagnostic procedure in Section 9-4. Use the error reference tables in Sections 9-2 and 9-3 to identify the most likely cause before changing anything.

CAUTION

Do not make changes to a production pipeline or Ontology while diagnosing an error. Work on a development branch. Changing production while under fault conditions can cascade failures to downstream users and applications.

9-2. PIPELINE BUILDER ERRORS

Error / Symptom	Likely Cause	Resolution Steps
"Column not found"	Source dataset schema changed; a column was renamed or removed upstream	Open the failing node. Identify the missing column. Check the source dataset schema in Pipeline Builder preview. Update the node to use the current column name or rename/add the column in the source.
Join output returns 0 rows	Type mismatch on join key, OR no common values exist between the two key columns	Type-cast both join key columns to the same type (e.g., both to string). Sample both datasets and visually verify that key values overlap. Confirm neither key column is all-null.
Join output row count exceeds either input dataset	Fan-out due to a one-to-many relationship on the right-side dataset	Add a deduplication step upstream of the join on the right-side dataset, OR aggregate post-join. COUNT(*) before and after each join step to trace where inflation occurs (see Pattern B-2).

Error / Symptom	Likely Cause	Resolution Steps
Pipeline fails with "type cast error"	Source column contains non-numeric text (e.g., "N/A", "--") in a field being cast to integer or date	Inspect 50 rows using the Pipeline Builder sample function. Add a filter node to remove or replace invalid values before the type cast step.
Pipeline fails with "null reference"	Required column is missing from the source dataset entirely, or a referenced column was deleted	Check the source dataset schema. Add a null-handling step (COALESCE or filter) for all required fields before they are consumed downstream.
Pipeline output is empty but no error is shown	A filter step is excluding all rows	Review all filter conditions in the pipeline. Use the pipeline sample function at the step immediately before each filter to verify data exists at that point.
Pipeline runs successfully but output looks wrong	Wrong join grain — fan-out inflating records or a left join silently dropping records	COUNT(*) at each stage (source, post-join, post-filter, output). Trace where the row count diverges from expectation.
Ontology write fails: "duplicate primary key"	Pipeline output contains multiple rows with the same primary key value	Add a deduplication step before the output node. Verify the primary key definition is correct and matches the Ontology Object Type PK.
Ontology write fails: "type mismatch"	A pipeline output column type does not match the Ontology property type it maps to	Open the Object Type in Ontology Manager. Check the property data type. Add a type cast in the pipeline to align the column type to the Ontology property definition before writing.
Dataset shows stale data despite pipeline appearing active	Scheduled pipeline run failed silently or schedule was not saved	Open the pipeline run history. Check the most recent run status and error log. Re-run manually. Confirm the schedule is saved and active on the production branch (not the dev branch).
"Pipeline scheduled but not running"	Schedule not saved, or pipeline is on a development branch, not the production branch	Confirm the pipeline is on the main/production branch. Open the Schedule panel and verify the schedule is active.
Source data volume drops unexpectedly	Source system stopped sending data, or an upstream pipeline failed	Check the source dataset row count against historical baseline. Verify the upstream connector or ingestion pipeline is running. Alert your Data Steward if a source feed appears to have stopped.

NOTE

The most common pipeline error is a silent one — the pipeline succeeds but produces incorrect output because of a grain mismatch or a fan-out join. Always COUNT(*) before and after joins. A pipeline that builds green is not the same as a pipeline that is correct.

9-3. WORKSHOP APPLICATION ERRORS

Error / Symptom	Likely Cause	Resolution Steps
Workshop table shows "No data"	Object Type has no records, OR an active filter is set to a non-existent value	Verify the Ontology Object Type has data (open in Quiver and confirm row count > 0). Reset all Workshop filters to their default state and retest.
Action button not visible to end users	User lacks Editor role, OR Action is not connected to a button widget	Open Workshop editor. Check the button widget's Action binding. Verify the user has been added to the correct project role (Viewer is sufficient to execute most Actions — confirm in Ontology Manager Action settings).
Filter widget not filtering the table or chart	Filter is not connected to the table or chart widget	Open Workshop editor. Select the table or chart widget. In the Widget Properties panel, find the filter binding section and connect the filter widget to the table/chart.
Dashboard loads slowly or times out	Object Type has a very large record count (1M+) with no pre-applied filters on the source	Add a default filter to the Workshop application (e.g., restrict to current fiscal year or current AOR). If the Object Type itself is too large, escalate to a SL 3 builder for optimization of the backing pipeline or Ontology query configuration.
Workshop application not visible in the Workshop catalog	App is not published, OR user is not in the Viewer group	Open the app in Workshop editor and click Publish. Then open the Share panel and add the user to the Viewer role. Confirm the user logs in fresh after the permission change.
Form field shows wrong options in a dropdown	Action dropdown is bound to the wrong property or the backing dataset for the options is stale	Open the Action in Ontology Manager. Check the source of the dropdown values. Re-run the backing pipeline if the options list is outdated.
Chart displays incorrect	Chart aggregation setting does not match the grain	Open the chart widget properties. Verify the aggregation type (SUM, COUNT, AVG) matches what the user needs. If

Error / Symptom	Likely Cause	Resolution Steps
aggregation	of the Object Type	the Object Type is already a summary, use COUNT or first-value rather than SUM.

9-4. ACCESS AND PERMISSIONS ERRORS

Symptom	Likely Cause	Resolution Steps
"You do not have permission to view this resource"	Your account does not have the required role for this project or dataset	Contact your unit Data Steward to request the appropriate role. Do not attempt to work around access controls. Document the request per your unit's access management SOP.
Cannot find a project in Compass	Project may be in a different namespace, or you do not have access	Search by exact project name. Ask your Data Steward which project your work should be in. Do not create a new project — new projects require C2DAO authorization.
Cannot merge or promote branch changes	You may not have the required role to merge, or peer review has not been completed	Confirm peer review is complete (Chapter 7). If your role is insufficient, ask your Data Steward or a senior builder to promote on your behalf.
Users cannot see published Workshop application	Users not added to Viewer role on the application or the backing Object Type	Open the Share panel of the Workshop application. Add the users or group to the Viewer role. Confirm Ontology Object Type access includes the same users.
Action executes but write fails silently	User has Viewer role but the Action requires Editor-level write permission on the backing dataset	Review Action configuration in Ontology Manager. Confirm the write target dataset access grants are correct for the intended user population. Escalate to Data Steward to adjust permissions.

9-5. GENERAL DIAGNOSTIC PROCEDURE

BLUF: When you encounter an error not identifiable from the tables above, follow this five-step procedure before escalating.

1. **Identify the layer.** Is the problem in the Pipeline (data not producing correctly)? The Ontology (objects not appearing or linked incorrectly)? Workshop (display or interaction failure)? Access (permissions error)? The layer determines where to look.
2. **Check the run log or browser console for the exact error message.** Do not rely on the symptom description alone. The error message often names the specific node, column, or operation that failed. Copy the exact error text before doing anything else.
3. **Sample the input data at the step where the error occurs.** Use the Pipeline Builder sample function on the node immediately upstream of the failure point. Verify the data shape (column names, types, sample values) matches what the downstream node expects.
4. **Compare input to output — count rows, spot-check values.** Run COUNT(*) at the input and output of each transform step. If row counts diverge unexpectedly, the problem is at that step. Spot-check 10–20 rows at the problem step before and after the transform.
5. **If unresolved in 15 minutes, escalate.** Contact your unit Data Steward or a SL 3 Advanced Builder. Provide: (a) the exact error message, (b) the pipeline or app name and environment (dev/main), (c) the layer where the error occurs, and (d) what you have already tried. Do not continue making changes while the issue is unresolved — each change makes the error harder to diagnose.

9-6. WHEN TO ESCALATE

Escalate to your unit Data Steward when: - You cannot diagnose the cause of a pipeline failure after 30 minutes. - An Ontology change you made broke a downstream application. - You are unsure whether a change requires a governance review. - Any security or data handling concern arises. - A source feed appears to have stopped (missing expected data).

Escalate to C2DAO when: - The Data Steward cannot resolve the issue. - A production system is down and affecting multiple units. - An AIP Logic or advanced feature is required (SL 3 scope). - A new data source connection is required (all new connectors require C2DAO authorization).

NOTE

Escalating early is not a sign of failure. Spending two hours on a problem that requires a SL 3 developer or a Data Steward access change is wasted time. The 15-minute threshold exists to protect operational tempo.

APPENDIX A — PRE-PUBLISH CHECKLISTS

BLUF: Complete the applicable checklist before every merge request, Ontology branch promotion, or Workshop application publication. These checklists are the SL 2 builder's quality gate. Do not skip steps.

Four checklists correspond to the four major build outputs: dataset ingestion, pipeline, Ontology Object Type, and Workshop application. A given build task may require you to complete more than one.

A-1. DATASET INGESTION CHECKLIST

Complete before publishing a newly ingested dataset for others to use.

- ☐ Row count matches source system (within 1% tolerance, or deviation documented with explanation)
- ☐ No unexpected null rates greater than 5% in required fields (document and accept with Data Steward approval if unavoidable)
- ☐ Column data types confirmed: no text in numeric fields, dates parsed correctly, no mixed-type columns
- ☐ Sample of at least 50 rows reviewed for data quality (spot-check values against known source records)
- ☐ Dataset name follows naming convention: `[DOMAIN]_[ENTITY]_[GRAIN]` (e.g., `MAINT_EQUIP_RECORD`, `PERS_SOLDIER_DAILY`)
- ☐ Dataset description field populated with: source system name, ingestion date, data owner name and unit, refresh frequency
- ☐ Classification marking applied correctly (UNCLASSIFIED / CUI as appropriate — when in doubt, treat as CUI and confirm with Data Steward)
- ☐ Access control configured on the principle of minimum necessary — not open to all Project members if the data is restricted
- ☐ Data Steward notified of new dataset availability; Data Steward has acknowledged and approved the data source
- ☐ Dataset placed in the correct folder tier: `raw/` for unprocessed source data; `staging/` for intermediate cleaned data; `curated/` for publication-ready data

WARNING

Never mark a dataset as curated until data quality checks are complete and Data Steward approval is on record. Downstream Object Types and Workshop applications depend on curated datasets being stable and trustworthy.

A-2. PIPELINE CHECKLIST

Complete before promoting a pipeline to production.

- ☐ Input row count documented (run the pipeline in Dev with SAMPLE to confirm the source is live and producing the expected volume)
- ☐ Output row count matches expected (no unexpected fan-out from joins, no unexpected data loss from filters)
- ☐ Join grain verified: COUNT(*) before and after each join step — output should not exceed the larger input unless fan-out is intentional and documented
- ☐ Null handling documented: COALESCE or explicit filter applied for all required fields before they flow into downstream steps
- ☐ Type casts explicit and tested: verify edge cases (e.g., cells containing "N/A", empty string, or leading zeros) against a sample
- ☐ Deduplication step included if source can contain duplicate records (document which record is kept and why — this is a business rule)
- ☐ Pipeline description field populated: inputs, transformations applied, output grain, data owner, refresh schedule
- ☐ Schedule configured and tested (if recurring): confirm first scheduled run completed successfully; confirm schedule is active on the production branch, not the dev branch
- ☐ Peer review completed: a second Builder or the Data Steward has reviewed the pipeline logic and signed off
- ☐ Changes promoted from a Dev branch — never edited directly in production
- ☐ On-failure notification configured (builder and team lead at minimum)
- ☐ Pipeline complexity is within SL 2 scope. If the pipeline requires multi-source deduplication, @incremental patterns, custom Python transforms, or complex error-handling logic, escalate to a SL 3 builder before proceeding.

CAUTION

A pipeline that builds without errors is not necessarily a correct pipeline. "Green" means the transform ran — it does not mean the output is accurate. Verify row counts and spot-check values at every stage.

A-3. ONTOLOGY CHECKLIST

Complete before an Object Type goes live on the production branch.

- ☐ Primary Key defined and verified unique in the backing dataset (zero duplicate PK values in pipeline output)
- ☐ Property names follow naming convention: PascalCase, no spaces, no generic names (e.g., `EquipmentId` not `id` or `column_1`)
- ☐ Property data types set correctly — **verify before publishing**; property types are immutable after downstream objects connect and changing them is a breaking change requiring a governance review
- ☐ All required Properties populated: no critical fields are always null in the preview
- ☐ Link Types defined for all relationships between Object Types — do not substitute ad-hoc foreign key properties for properly configured Link Types
- ☐ At least one Action defined for any Object Type that operators will edit or update; Action tested with test data (write-back confirmed working)
- ☐ Object Type description populated: what it represents, which data source backs it, who owns it, and the refresh frequency
- ☐ Downstream consumers identified: any existing Workshop applications or analyses that will be affected by this change have been identified and tested post-change
- ☐ Changes checked in with descriptive commit messages (not "update" or "fix")
- ☐ Merge request includes: description of changes made, test results, and explicit notation of any breaking changes
- ☐ Merge request assigned to a reviewer other than yourself
- ☐ Tested with a Viewer-role user account (not the builder's Editor account) to confirm Actions execute correctly at the intended permission level
- ☐ Ontology design is within SL 2 scope. If design requires many-to-many links, multi-step Actions, derived properties with complex logic, or coalition-facing access, escalate to SL 3 before proceeding.

A-4. WORKSHOP APPLICATION CHECKLIST

Complete before publishing a Workshop application for operator use.

- ☐ All filter widgets connected to all relevant tables and charts (test each filter individually — a disconnected filter silently does nothing)
- ☐ Default filter values set to reasonable defaults; blank default often returns all records or no records — both are usually wrong
- ☐ All Actions tested with a Viewer-role user account (Editors see different permissions; testing as an Editor does not validate the operator experience)
- ☐ Application tested with realistic data volume (not just a 10-row test dataset — verify performance at production scale)
- ☐ Performance baseline noted: page load under 5 seconds for the target user base at expected data volume; escalate to SL 3 if threshold is exceeded
- ☐ Application name follows naming convention and is plain English, unit-appropriate
- ☐ Application header includes: title, classification/handling marking, owning unit, Data Steward POC
- ☐ Application description populated: purpose, intended users, data freshness (how current is the underlying data?)
- ☐ All charts have descriptive titles
- ☐ All filters have labels; Metric Tiles have labels and correct aggregation types
- ☐ Form submission tested with test data; write-back confirmed working
- ☐ Application renders correctly in Preview mode at 1920x1080
- ☐ Access control configured: only authorized users in the Viewer group; Link Sharing is OFF unless C2DAO specifically authorized
- ☐ Branch merged to main before publishing; application not published from a dev branch
- ☐ Published shared link tested from a user account outside the builder's session
- ☐ Users notified of application publication through official channels (unit email, SharePoint announcement, or similar)

APPENDIX B — COMMON PIPELINE BUILDER PATTERNS

BLUF: The following patterns address the most common USAREUR-AF data pipeline scenarios. Each pattern is reusable — adapt it to your specific data and grain requirements.

> **NOTE**
>
> The design patterns in this appendix are SL 2 level — they use Pipeline Builder without code. As your data products grow in complexity, some patterns will need to evolve into SL 3 designs. If a pattern requires multi-step Actions, complex Link Type logic, @incremental transforms, or custom Python/SQL code, refer to TM-30, Chapter 3 (Advanced Pipeline Builder) before proceeding.

B-1. DEDUPLICATION (KEEP FIRST OCCURRENCE)

Use case: Source data contains duplicate records for the same entity (e.g., multiple status entries for the same equipment ID in a GCSS-A extract, or multiple SITREP rows for the same unit on the same report date).

When to apply: After ingestion, before any join step or Ontology write.

Procedure:

1. Add a Sort node. Sort by entity ID (ascending) and by timestamp or sequence column (descending if you want the most recent, ascending if you want the earliest).
2. Add a Deduplicate node immediately after the Sort node. Configure it to deduplicate on the entity ID column. The Deduplicate node retains the first row it encounters per group — the Sort order determines which row is "first."
3. Run the pipeline and verify: output row count should equal the count of distinct entity IDs.
4. Document the deduplication logic in the pipeline description (e.g., "Keeps the most recent record per bumper_number as determined by maint_date descending").

```
[Source: gcss_a_maint_raw]
        |
[Sort: bumper_number ASC, maint_date DESC]
        |
[Deduplicate: on bumper_number — retains most recent record]
        |
[Output: gcss_a_maint_deduped_staging]
```

NOTE

Which occurrence you keep is a business rule, not a technical default. "Most recent" is common but not always correct — confirm with the data owner before choosing a deduplication strategy.

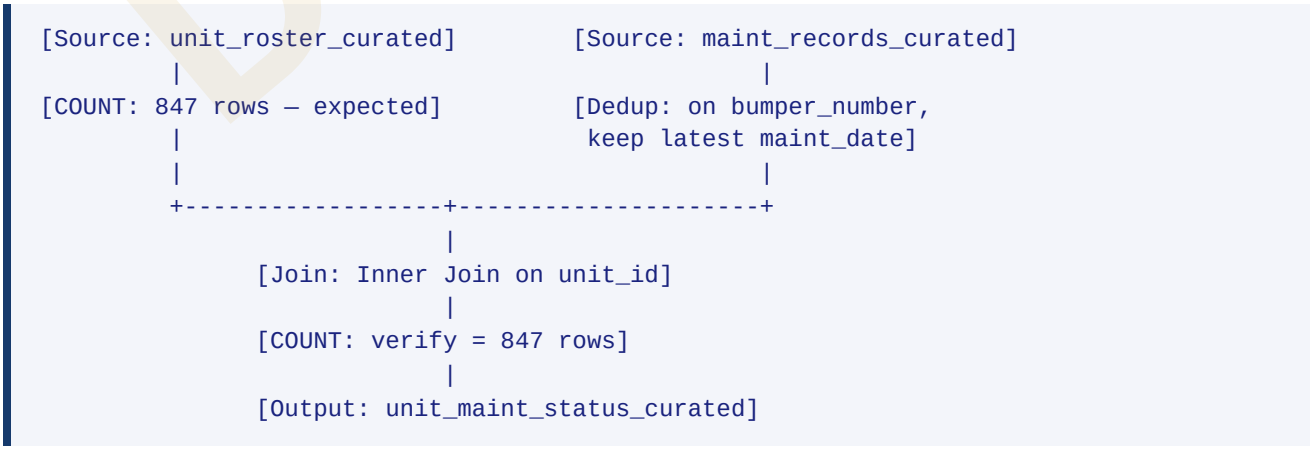
B-2. JOIN WITH FAN-OUT DETECTION

Use case: Joining two datasets on a foreign key where the right-side dataset may have multiple matching rows per key value (e.g., joining a unit roster to a maintenance table where one unit can have many equipment items).

When to apply: Any time you perform a join where you are not certain the right-side dataset has exactly one row per join key value.

Procedure:

1. COUNT(*) the left-side input dataset before the join. Record that number.
2. COUNT(*) the right-side input dataset before the join. Record that number.
3. Perform the join.
4. COUNT(*) the join output. Compare to step 1.
5. If output count equals left-side input count: no fan-out. Proceed.
6. If output count is greater than left-side input count: fan-out has occurred.
7. If fan-out occurred: add a deduplication or aggregation step on the right-side dataset before the join. Re-run and recheck counts.



CAUTION

Fan-out silently inflates row counts and produces incorrect aggregations downstream. A SUM that runs against fan-out data will overcount. This is one of the most common and consequential pipeline errors.

B-3. TYPE CAST AND NULL HANDLING

Use case: Source data has mixed types or null values in required fields (e.g., an equipment ID column that contains "N/A" strings alongside numeric IDs, or a date column with blank cells).

When to apply: Immediately after ingestion, before any join or aggregation that depends on the affected columns.

Procedure:

1. Add a Sample step and inspect 50–100 rows of the source data. Identify columns with mixed types or unexpected values.
2. Add a Filter node to remove or isolate rows with invalid values in required fields (e.g., filter out rows where `equipment_id = 'N/A'` or where `report_date IS NULL`).
3. Add a Type Cast node. Cast the column to the target type (text → integer, text → date, etc.).
4. Add a COALESCE step (calculated column or null-replace node) to replace any remaining nulls with a documented default value.
5. Add a post-transform Sample to verify the output contains only valid typed values.

```
[Source: equip_status_raw]
       |
[Filter: Remove rows where equipment_id = 'N/A' OR equipment_id IS NULL]
       |
[Type Cast: report_date (text) → date; equipment_id (text) → integer]
       |
[Calculated Column: COALESCE(status_code, 'UNKNOWN') AS status_code]
       |
[Sample: verify types and values before output]
       |
[Output: equip_status_staging]
```

NOTE

Always handle nulls BEFORE type casting. Casting a null can fail or produce unexpected results depending on the source data format. The order is: filter invalid values → cast types → coalesce remaining nulls.

B-4. ROLLUP AND AGGREGATION (CHANGING GRAIN)

Use case: You need a summary dataset from row-level data — for example, counting open work orders by unit from a row-per-work-order source, or summing personnel strength by company from a row-per-Soldier source.

When to apply: When the Object Type or Workshop widget requires summary-level data, not individual records.

Procedure:

1. Identify the target grain (e.g., one row per unit per day).
2. Add a Group By node. Select the columns that define the target grain (e.g., `unit_id`, `report_date`).
3. Configure the aggregation metrics: SUM for quantities, COUNT for totals, MAX or MIN for extremes, AVG for rates.
4. Run the pipeline and spot-check: pick 2–3 groups from the output and manually verify the aggregated value against the source sample.
5. Document the grain in the output dataset description.

```
[Source: maint_records_curated]  (one row per work order)
       |
[Group By: unit_id, report_date
 Aggregations: COUNT(*) as open_work_orders,
               SUM(estimated_repair_hours) as total_repair_hours]
       |
[Output: maint_summary_by_unit_daily_curated]  (one row per unit per day)
```

NOTE

Document the grain change in the output dataset description. Downstream users must know this is a summary, not row-level data. If a user needs row-level detail later, they must return to the pre-aggregation dataset.

B-5. STANDARD SITREP INGESTION PATTERN

Use case: Daily SITREP feed arrives as a structured file or shared dataset. Clean, deduplicate, and load to curated layer for Ontology backing.

Procedure:

1. Connect to the authorized source connector for the SITREP feed (SharePoint file, SFTP, or shared dataset — coordinate with Data Steward for the correct connector).
2. Add a Filter node to remove rows where `unit_id IS NULL`.
3. Add a Select Columns node to retain only the required fields.
4. Add a Rename Columns node to align column names to the curated naming convention.
5. Add a Deduplicate node on `unit_id + report_date` to keep only one record per unit per reporting period (keep most recent).
6. Run the pipeline and verify: output row count should equal the number of reporting units.

```
[Source: SITREP Feed]
       |
[Filter: Remove rows where unit_id IS NULL]
       |
[Select Columns: unit_id, unit_name, report_date, status_code, remarks]
       |
[Rename Columns: status_code → status, remarks → sitrep_text]
       |
[Deduplicate: on unit_id + report_date — keep most recent]
       |
[Output: sitrep_feed_curated]
```

Schedule: Daily at 0200 UTC (before morning staff meetings).

> **NOTE**
>
> Alert your Data Steward if the output row count drops below the historical average — this indicates one or more units failed to report. Do not assume a low row count is normal.

B-6. UNION OF MULTIPLE SOURCES

Use case: The same entity type is reported by multiple source systems with different schemas (e.g., readiness data from III Corps, V Corps, 21st TSC, 7th ATC, 10th AAMDC, 56th MDC-E, and SETAF-AF each arriving via separate feeds; or MTOE units from one system and augmentation force units from another).

When to apply: When you need a single consolidated dataset representing all instances of an entity regardless of source.

Procedure:

1. Ingest each source dataset separately.

2. For each source, add a Rename/Select node to normalize the schema: ensure column names and data types match exactly across all sources. This step is mandatory — do not skip it.
3. Add a `source_system` Calculated Column to each branch with the source name as a literal value (e.g., `'VCORPS'`, `'21TSC'`, `'7ATC'`). This preserves data provenance in the output.
4. Connect all branches to an Append (Union) node.
5. Add a Deduplicate node after the Append if sources can overlap (i.e., the same entity appears in more than one source).
6. Verify: output row count should equal the sum of all branch row counts, minus any cross-source duplicates removed in step 5.

```
[Source: VCORPS Feed]   [Source: 21TSC Feed]   [Source: 7ATC Feed]
         |                       |                      |
[Normalize schema +     [Normalize schema +    [Normalize schema +
 source_system='VCORPS'] source_system='21TSC']  source_system='7ATC']
         |                       |                      |
         +-----------+-----------+----------------------+
                     |
           [Append (Union): combine all rows]
                     |
           [Deduplicate: on unit_id + report_date — if sources can overlap]
                     |
           [Output: readiness_consolidated_curated]
```

CAUTION

Never union two datasets without first confirming column type alignment. A column that is `text` in one source and `integer` in another will fail silently or cause type errors downstream. Normalize types in each branch before the Append node.

B-7. LOGISTICS MAINTENANCE TRACKING PATTERN

Use case: GCSS-A maintenance data needs to be joined with unit assignment data to show which unit owns each item with open work orders.

Procedure:

1. Ingest the GCSS-A maintenance source dataset.
2. Add a Filter node to retain only records with active work orders (`status_code IN ('OPEN', 'IN_PROG')`).
3. Ingest the unit assignment curated dataset.

4. Standardize the join key (bumper number) on both sides: verify case, leading zeros, and spacing match exactly. Add a Rename or Calculated Column node to normalize formatting if they differ.
5. Perform an Inner Join on the bumper number column.
6. Output to curated layer.

```
[Source: gcss_a_maint_raw]          [Source: unit_assignment_curated]
         |                                        |
[Filter: status_code IN                 [Select: unit_id, bumper_number,
 ('OPEN', 'IN_PROG')]                    unit_name, home_station]
         |                                        |
         +------------------------+---------------+
                                  |
                  [Join: Inner Join on bumper_number]
                                  |
                  [Output: maint_open_work_orders_curated]
```

CAUTION

Bumper number formatting inconsistencies (case, leading zeros, hyphens) are the most common cause of zero-row output on this join. Always sample both datasets and visually compare key values before running a full join.

Q1 2026 PLATFORM UPDATES

The following Palantir Foundry updates affect SL 2 content. Builders should familiarize themselves with these changes.

OBJECT VIEWS — GENERAL AVAILABILITY

Object Views reached GA in Q1 2026. Key improvements relevant to SL 2 builders:

- **Section-based layouts:** Object Views now support configurable sections (grouping related properties under collapsible headers) in addition to the flat property list. Builders can organize properties into operational groupings (e.g., "Identity," "Status," "Location") for improved readability.
- **Conditional visibility:** Properties can be shown or hidden based on the value of another property. Example: a "Maintenance Notes" section appears only when equipment status is "NMC."
- **Linked Object previews:** Object Views can now display inline previews of linked objects (e.g., a `MaintenanceRecord` view can show the linked `UnitStatus` object's key properties without navigating away).
- **Impact on TASK 4-3:** The Object View configuration procedure in this manual remains valid. The new section-based layout is an optional enhancement — builders should adopt it for Object Types with more than 10 displayed properties.

PIPELINE BUILDER ENHANCEMENTS

- **Live row-count preview:** Pipeline Builder now displays estimated row counts at each node in the pipeline graph without requiring a full build. Use this to detect fan-out or unexpected filtering earlier in the build process.
- **Improved error messages:** Transform node errors now include the specific column name and sample values that caused the failure, reducing diagnostic time.
- **Template pipelines:** Builders can now save a pipeline configuration as a template and instantiate copies for similar data sources. This supports the standard ingestion patterns in Appendix B.

FOUNDRY BRANCHING IMPROVEMENTS

- **Branch comparison view:** Before submitting a merge request, builders can now view a side-by-side diff of all resource changes (pipeline logic, Ontology edits, Workshop layout) between the dev branch

and production. This supports the review process described in TASK 7-3.

- **Auto-conflict detection:** Foundry now flags merge conflicts before submission, preventing failed merges that previously required C2DAO intervention to resolve.

GLOSSARY

Terms are defined in plain English with USAREUR-AF operational context where applicable. Foundry-specific terms reflect SL 2 no-code usage; SL 3 and advanced usage is noted where the term has a broader meaning.

Action — A configured, form-based operation in the Foundry Ontology that allows authorized users to write data back to a backing dataset. Examples: submitting a SITREP update, marking a maintenance record complete, or updating a unit status field. Actions are configured in Ontology Manager and invoked through Workshop form widgets or buttons. SL 2 Actions are single-step. Multi-step Actions with conditional routing, approval chains, or sequential writes are SL 3 scope.

Aggregation — A mathematical operation that reduces multiple rows to a single summary value. Common aggregation functions: SUM (total a numeric column), COUNT (count rows), AVG (average), MAX (highest value), MIN (lowest value). Aggregation always changes the grain of a dataset. Document the grain change whenever you aggregate. See Pattern B-4.

AOR — Area of Responsibility. The geographic and functional area within which a command has authority to act. USAREUR-AF's AOR covers the European theater. An AOR filter is a common default filter applied in Workshop applications to restrict data to records relevant to a specific unit's geographic responsibility.

Branch (Dev/Production) — An isolated environment copy in Foundry where a builder makes and tests changes without affecting the live system. Development (dev) branches are the builder's workspace. The production branch (typically called `main`) is what all users see. Changes move from dev to production via a merge request and peer review. SL 2 builders never edit the production branch directly.

Builder — A Foundry user with Editor-level access who creates and modifies pipelines, Ontology configurations, and Workshop applications. Builders have elevated privileges compared to Operators (SL 1 users). Builder access requires chain-of-command approval and completion of this manual.

Calculated Column — A transform step in Pipeline Builder that creates a new column by applying a formula or expression to existing columns. Examples: concatenating two fields, computing a derived value, or applying COALESCE to replace nulls. No code required at SL 2 level — Pipeline Builder provides a formula editor UI.

Cardinality — The description of how many records in one dataset or Object Type relate to records in another. Key cardinality types: One-to-One (one readiness record per unit per day), One-to-Many (one unit has many equipment items), Many-to-Many (many soldiers can be assigned to many tasks). Cardinality determines how to configure Link Types and how to handle joins to avoid fan-out. See Section 9-2 and Pattern B-2.

Classification Marking — The required security designation applied to every dataset and application on MSS. Standard markings: UNCLASSIFIED, CUI (Controlled Unclassified Information), SECRET. Marking must be applied before any dataset is published or shared. When in doubt, treat data as CUI and confirm with your Data Steward before publishing. Incorrect marking is a reportable security incident.

Coalesce — A function that returns the first non-null value from a list of inputs. In Pipeline Builder, COALESCE is used to replace null values with a documented default (e.g., `COALESCE(status_code, 'UNKNOWN')`). Always apply COALESCE after filtering invalid values and before type casting. See Pattern B-3.

Contour — The Foundry interactive analysis tool. Allows builders and analysts to build table, chart, and pivot views of datasets and Object Types. Analyses are saved as reusable, shareable resources. Use Contour for ad-hoc data exploration and quality-checking pipeline outputs. Contour reads from both datasets and Ontology Object Types.

Data Steward — The unit-level accountable official for a specific data domain. First escalation point for all data governance questions. Must approve new data source connections, access changes, and classification decisions. The Data Steward is not optional — they are the gateway between your build activity and the broader data governance chain.

Dataset — A tabular data resource in Foundry, organized as rows and columns. Datasets exist in three tiers: raw (unprocessed, direct from source), staging (cleaned but not yet validated), and curated (publication-ready, schema-stable). Ontology Object Types are backed by curated datasets only. See also: Backing Dataset, Raw Dataset, Staging Dataset, Curated Dataset.

Deduplication — The process of removing duplicate rows from a dataset, retaining only one record per unique entity (or entity + time period). Deduplication is a business rule decision — which record you keep must be documented and justified. See Pattern B-1. Common causes of duplicate data: source systems that send repeated records on each extract, multiple feeds reporting the same event, or fan-out from a poorly constructed join.

ETL (Extract, Transform, Load) — The three-step process of pulling data from a source (Extract), cleaning or reshaping it (Transform), and writing the result to a destination (Load). Pipeline Builder is MSS's no-code ETL tool. Understanding ETL is the conceptual foundation for all pipeline work in this manual.

Fan-Out — A pipeline defect that occurs when a join inflates the row count beyond what is expected, because one side of the join has multiple matching rows per key value. Fan-out silently corrupts downstream aggregations. Always COUNT(*) before and after joins to detect fan-out. See Section 9-2 and Pattern B-2.

Filter — A transform step (in Pipeline Builder) or a UI element (in Workshop) that restricts data to rows matching specific conditions. In Pipeline Builder, filters reduce the dataset at a given pipeline step. In Workshop, filter widgets allow operators to interactively narrow the displayed data. All Workshop filter widgets must be explicitly connected to the tables and charts they are intended to control.

Foreign Key — A column in one dataset that contains values matching the primary key of another dataset. Foreign keys are used to configure Link Types between Object Types in the Ontology and to perform join operations in Pipeline Builder. Foreign key values must match the format and type of the primary key they reference — type or format mismatches are a common cause of zero-row join output.

Foundry (Palantir Foundry) — The enterprise data platform on which MSS is built. Foundry provides the full stack: data ingestion (connectors), data processing (Pipeline Builder), semantic modeling (Ontology), analysis (Contour, Quiver), and application development (Workshop). All tools described in this manual are components of Palantir Foundry deployed as MSS by USAREUR-AF.

Grain — The level of detail represented by a single row in a dataset. Defining and documenting grain is required for every curated dataset and Ontology Object Type. Example grains: one row per unit per report date; one row per work order; one row per Soldier per fiscal year. Grain determines how joins are structured and whether aggregation is needed before an output step. Grain must be documented in the dataset description.

Join — A transform step in Pipeline Builder that combines two datasets by matching rows where a specified key column has the same value in both datasets. Join types: Inner Join (only rows with matching keys in both datasets), Left Join (all rows from the left dataset, matched where possible), Right Join (all rows from the right dataset). SL 2 scope: Inner and Left joins on a single key column. Multi-key or complex joins are SL 3 scope.

Link Type — A configured relationship between two Object Types in the Foundry Ontology. Examples: a `MaintenanceRecord` linked `ownedBy` a `UnitStatus`; a `SoldierReadiness` linked `assignedTo` a `UnitRoster`. Link Types are configured in Ontology Manager using the foreign key column on one Object Type matched to the primary key of another. SL 2 scope: one-to-one and one-to-many Link Types. Many-to-many Link Types with junction datasets are SL 3 scope.

Null — The absence of a value in a column. Null is not the same as zero (for numbers) or empty string (for text). Null values in required fields cause pipeline failures, incorrect aggregations, and broken join keys. Handle nulls explicitly: filter them out if they represent invalid records, or replace them with a documented default using COALESCE if a default is acceptable. Always handle nulls before type casting.

Object Type — A defined class of entities in the Foundry Ontology. Each Object Type represents a real-world thing the system tracks: a unit, an equipment item, a maintenance record, a personnel readiness entry. Object Types are named in PascalCase singular noun format (e.g., `UnitStatus`, `MaintenanceRecord`). Object Types are backed by curated datasets and exposed to users through Workshop applications and Contour analyses.

Ontology — The semantic layer of Foundry. The Ontology defines what the data means: Object Types (what things exist), Link Types (how they relate), and Actions (what users can do). Workshop applications read from the Ontology, not directly from datasets. The Ontology is the bridge between raw data and user-facing applications. Changes to the Ontology affect all downstream applications — treat them as breaking changes until proven otherwise.

Pipeline — A configured sequence of data transform steps in Pipeline Builder, connecting a source dataset to an output dataset via a series of nodes (filters, joins, renames, aggregations, etc.). Pipelines are the primary ETL mechanism in MSS. Each pipeline should have one clear input, a documented set of transforms, and one curated output dataset. Pipeline logic must be peer-reviewed before production promotion.

Primary Key — The column (or combination of columns) that uniquely identifies each row in a dataset and each object in an Object Type. Primary keys must be unique (no duplicates) and non-null. Verify primary key uniqueness before every Ontology write. Duplicate primary keys will cause the Ontology write to fail or produce unpredictable object behavior.

Project — The top-level organizational container in Foundry Compass. Projects hold all related datasets, pipelines, Ontology resources, and Workshop applications for a specific mission or unit. New projects require C2DAO authorization. Do not create a new Project when you cannot find yours — search for it and contact your Data Steward.

Property — An attribute of an Object Type, analogous to a column in a table. Each property maps to a column in the backing dataset. Properties are configured in Ontology Manager. Display names must be plain English (not raw column names). Data types are set at configuration time and are effectively immutable once downstream objects connect to the Object Type — verify types before publishing.

Quiver — The Foundry quick dashboard builder. Assemble charts and metric tiles into a simple, shareable page without Workshop's full widget framework. Quiver is well-suited for quick data quality checks, briefing-support views, and ad-hoc analysis. Quiver does not support Actions or interactive filter bindings between widgets — use Workshop for operator-facing applications.

Row Count — The number of records in a dataset or pipeline output at a given step. Row count is the primary sanity check for all pipeline work. Document the expected row count for every curated dataset output. Verify row count before and after every join, filter, and aggregation step. Unexpected row count changes are the most reliable early indicator of a pipeline defect.

Schema — The structure of a dataset: the set of column names, their data types, and their order. Schema stability is required for curated datasets — downstream Object Types and pipelines break if a curated dataset's schema changes without coordination. Never rename, remove, or change the type of a column in a curated dataset without notifying all downstream consumers first.

Type Cast — A transform step that converts a column from one data type to another (e.g., text → integer, text → date). Type casting is required when source data delivers values in the wrong type for downstream use. Always filter invalid values before type casting, and always apply COALESCE after casting to handle any remaining nulls. See Pattern B-3.

Union — A transform step (called Append in Pipeline Builder) that combines rows from two or more datasets into a single dataset. All input datasets must have identical column names and data types before the Union node. Add a source provenance column before unioning to track which source each row came from. See Pattern B-6.

Workshop — The Foundry drag-and-drop application builder. Builders assemble widgets (tables, charts, filters, forms, metric tiles, maps) into interactive applications for end users. Workshop reads from the Ontology — it does not read directly from datasets. No code is required for SL 2-level applications. Workshop applications are published and access-controlled through the Share panel. All filter widgets must be explicitly connected to the widgets they control.

TM 20-MSS-BLD

HEADQUARTERS, UNITED STATES ARMY EUROPE AND AFRICA, Wiesbaden, Germany

2026

By order of the Commander, United States Army Europe and Africa.