LESSON PLANS

SL 2

USAREUR-AF

LESSON PLAN OUTLINES — SL 2 BUILDER

Maven Smart System (MSS) Training Program

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

26 MARCH 2026

DRAFT — UNOFFICIAL — NOT FOR OPERATIONAL USE

LESSON PLAN OUTLINES — SL 2 BUILDER

MAVEN SMART SYSTEM (MSS) TRAINING PROGRAM

USAREUR-AF Operational Data Team — C2DAO

> **NOTE**
>
> These are lesson plan outlines — abbreviated LP format for instructor preparation. Use the full LP template (`LP_TEMPLATE.md`) to expand any block into a complete lesson plan. For complete lesson plan examples see `TM10/TM10_LESSON_PLANS.md` .

Field	Value
Course	SL 2 Builder
Duration	5 days (40 hours)
Version	1.0 — March 2026

AUTHORITATIVE REFERENCES

Publication	Title	Relevance
AR 350-1	Army Training and Leader Development	Master regulation for Army training policy; governs lesson plan standards and instructional requirements
FM 7-0	Training	Unit training management procedures; context for how SL 2 builder training integrates with unit training plans

DAY 1 — PROJECT FUNDAMENTALS AND FILE INGESTION

BLOCK 1 — Project Creation

Hours: 1.5 | **Method:** Lab | **Time:** 0830–1000

Purpose: Establishes the C2DAO naming and structure standard before any data is touched. A project created incorrectly will fail data steward promotion review.

TLO: Given C2DAO naming conventions, the trainee will create a correctly named, marked, and structured Foundry project — to standard, independently.

Key Delivery Notes: - Walk through the naming convention standard from TM-20 Ch 2: `[UNIT]_[DESCRIPTION]_[ENV]` format. Write it on the board for the whole day. - Show a badly-named project before showing a correctly-named one. Make the contrast memorable. - Folder structure: Datasets, Pipelines, Ontology, Applications. Create all four folders — even if empty. This prevents chaos later. - Classification marking: show where to set it on project creation. UNCLASSIFIED in training; trainees must know where to set it for production. - Demo, then students create their own. Confirm naming before moving on — fix non-compliant names now, not on Day 4.

Student Activity: Create a Foundry project following C2DAO naming conventions. Create all four required folders. Screenshot the folder structure for self-review.

Common Errors:

Error	Action
Spaces or special characters in project name	Catch before trainee creates datasets inside it
Missing classification marking	Fix before moving on — evaluator will check
Skipping folder structure	Ensure all four folders are created in this block

Assessment: Evaluated in Practical Exercise Task 1. Evaluator checks naming convention, classification marking, and folder structure.

BLOCK 2 — File Ingestion

Hours: 0.75 | **Method:** Lab | **Time:** 1015–1100

Purpose: Ingestion is the entry point for all data in Foundry. Trainees must be able to upload a file, inspect the inferred schema, and identify data quality issues before building any pipeline.

TLO: Given a CSV file, the trainee will ingest it into a Foundry project, confirm row count matches the source, and identify at least one type mismatch or null pattern in the schema.

Key Delivery Notes: - Upload a provided CSV file via drag-and-drop or file browser. Show both methods. - After upload, walk through the schema preview: column names, inferred types, sample values. - Foundry infers types — it may get them wrong (date column stored as text, numeric column with trailing spaces). Trainees need to see this, not just upload and move on. - Confirm row count: source file row count should match the ingested dataset row count minus the header.

Student Activity: Upload the provided CSV file. Verify row count. Identify at least one column where the inferred type appears incorrect.

Common Errors: Not verifying row count after ingestion; accepting inferred types without checking (particularly date columns stored as strings).

Assessment: Evaluated in Practical Exercise Task 2. Row count verification is a tested step.

BLOCK 3 — Dataset Explorer: Column Profiling

Hours: 1.0 | **Method:** Lab | **Time:** 1100–1200

Purpose: QC habits built before the pipeline prevent errors that are much harder to diagnose after.

TLO: Given an ingested Foundry dataset, the trainee will use the Dataset Explorer to profile column distributions, identify null rates, and document at least two data quality observations.

Key Delivery Notes: - Open Dataset Explorer for the ingested dataset. Walk through: column list, null percentage bar, distinct values count, distribution histogram (numeric columns). - Key habit: check null rate for every column you plan to use in a pipeline. A null rate of 20% on `c_rating` means 20% of downstream objects will have empty ratings — you need a plan for that before the pipeline runs. - Show the "preview" of top 10 rows alongside the column profile — the two views complement each other. - Have trainees document findings before moving to Block 4.

Student Activity: Profile all columns in the ingested dataset. Document: null rate for each, type anomalies, any unexpected distinct values.

Assessment: Supports pipeline quality in Practical Exercise Task 3. Evaluator may ask how the trainee verified data quality before building.

BLOCK 4 — Pipeline Builder Orientation

Hours: 2.0 | **Method:** Lab | **Time:** 1300–1500

Purpose: Trainees need a mental model of the Pipeline Builder canvas before adding steps. This block is orientation — not a complete pipeline build.

TLO: Given the Pipeline Builder canvas, the trainee will identify the step library, configure input and output dataset nodes, add a single step, and describe the data flow from input through step to output.

Key Delivery Notes: - Open Pipeline Builder from inside the Foundry project. Canvas orientation: input node (left), step nodes (center), output node (right). - Step library: show how to search for a step by type. Common steps used this week: Filter, Rename, Cast, Calculated Column, Join, Group By, Ontology Write. - Connect input dataset to a Filter step to output dataset — just to demonstrate the connection mechanic. Do not configure the filter yet (that is Day 2). - Run the pipeline with zero transformation (pass-through) and confirm output dataset populates.

Student Activity: Build a pass-through pipeline (input → no-op filter → output). Run it. Verify output row count equals input row count.

Common Errors:

Error	Action
Disconnected nodes (step not wired to both input and output)	Pipeline won't run; check connections
Output dataset path not set	Pipeline runs but output is lost; always configure the output path

Assessment: Foundation for all subsequent labs. Not directly evaluated but a prerequisite for every other pipeline task.

BLOCK 5 — C2DAO Naming Conventions Discussion

Hours: 0.5 | **Method:** Discussion | **Time:** 1500–1530

Purpose: Naming convention is the governance standard that makes data products maintainable. Deviations fail data steward promotion review.

Key Delivery Notes: - Reference `NAMING_AND_GOVERNANCE_STANDARDS.md`. Walk through naming patterns for: datasets, pipelines, Object Types, Workshop applications. - Have trainees look at their Block 1 project names and Block 2 datasets — do they follow the convention?

Student Activity: Review their own Block 1 project and Block 2 dataset names against the standard. Correct any deviations now.

BLOCK 6 — Individual Practice

Hours: 1.5 | **Method:** Lab | **Time:** 1530–1700

Purpose: Reinforce Blocks 1–4 with a second independent repetition.

TLO: The trainee will independently create a second project, ingest a provided dataset, and confirm naming compliance — without instructor assistance.

Key Delivery Notes: - Different dataset than Block 2. Instructor circulates and observes; does not assist unless there is a system error. - Trainees who are confident may help each other on the naming convention question — but not on the system steps.

Assessment: Self-assessed. Instructor notes who needed assistance for Day 2 support awareness.

DAY 2 — PIPELINE BUILDER: CLEAN AND TRANSFORM

BLOCK 7 — Pipeline: Filter, Rename, CAST

Hours: 2.0 | **Method:** Lab | **Time:** 0830–1030

Purpose: The three most-used cleaning steps. Every pipeline starts here.

TLO: Given a raw dataset with rows to exclude and columns to rename and type-cast, the trainee will build a pipeline with Filter, Rename, and CAST steps that produces a correctly typed, correctly named output.

Key Delivery Notes: - Filter: condition syntax — `column_name IS NOT NULL`, `c_rating IN ('C1', 'C2')`, `date_col >= DATE('2025-01-01')`. Add multiple conditions with AND/OR. - Rename: the output column name is what appears in all downstream steps and the Ontology. Get it right here. - CAST: explicitly type `string_date_col` as DATE. Show what happens when CAST fails (type mismatch error in pipeline logs). - Run after each step: don't add 5 steps and then run. Run incrementally — know which step introduced an error.

Student Activity: Build a 3-step pipeline (Filter → Rename → CAST) on the Day 1 dataset. Run after each step. Verify final output has correct types and only filtered rows.

Common Errors:

Error	Detail
Filter on a NULL-containing column using `!=` instead of `IS NOT NULL`	`NULL != 'C1'` does NOT return nulls; must explicitly filter nulls separately
CAST on a column that contains non-castable values	Pipeline error; review source data for anomalies before casting

Assessment: Foundation for Practical Exercise Task 3. Evaluator verifies pipeline runs to completion without error.

BLOCK 8 — Pipeline: Calculated Columns

Hours: 1.25 | **Method:** Lab | **Time:** 1045–1200

Purpose: Calculated columns derive new data from existing columns — string manipulation, conditional logic, null handling. This is where raw data becomes operationally meaningful.

TLO: Given a dataset, the trainee will add calculated columns using string functions, conditional IF/CASE logic, and COALESCE for null handling — producing a typed output column.

Key Delivery Notes: - String functions: `UPPER(col)`, `CONCAT(col1, ' - ', col2)`, `SUBSTRING(col, start, length)`. - Conditional logic: `IF(c_rating IN ('C3', 'C4'), 'NOT_READY', 'READY')`. CASE WHEN for multi-condition logic. - COALESCE: `COALESCE(last_maintenance_date, DATE('2000-01-01'))` — replace nulls with a default. Essential for columns used in downstream joins. - Always name the calculated column descriptively following the naming convention. `calc_1` is not acceptable.

Student Activity: Add three calculated columns: (1) a string concatenation field, (2) a CASE WHEN readiness flag, (3) a COALESCE null-fill on a date column.

Common Errors:

Error	Detail
Referencing a renamed column by its original name	Use the post-Rename column name in calculated columns
Forgetting to name the output column	Unnamed calculated columns use a default that violates naming conventions

Assessment: Evaluated in Practical Exercise Task 3 (computed columns in the clean pipeline).

BLOCK 9 — Pipeline: Date and Time Functions

Hours: 2.0 | **Method:** Lab | **Time:** 1300–1500

Purpose: Date arithmetic is the most failure-prone area in Pipeline Builder. Mastery of this block prevents the most common practical exercise failure mode.

TLO: Given a dataset with date columns, the trainee will build pipeline steps using DATEDIFF, DATE_TRUNC, and CURRENT_DATE — testing results against known-answer records to validate output.

Key Delivery Notes: - Date column types must be DATE or TIMESTAMP — not STRING. If the source has string dates, CAST before using date functions (reinforce Block 7). - DATEDIFF: `DATEDIFF('day', start_date, end_date)` — result is number of days between dates. Note: argument order matters and varies by Foundry version. Test with a known record before relying on it. - DATE_TRUNC: `DATE_TRUNC('month', date_col)` — useful for monthly aggregations. - CURRENT_DATE:

`DATEDIFF('day', last_maintenance_date, CURRENT_DATE())` — this is a live calculation; it changes every day the pipeline runs. - Known-answer test: provide trainees a record with known dates. They calculate DATEDIFF manually, then verify the pipeline output matches.

Student Activity: Build a calculated column for "days_since_last_update" using DATEDIFF. Test against 3 provided known-answer records. All 3 must match.

Common Errors:

Error	Detail
Argument order in DATEDIFF reversed	Produces negative numbers; easy to catch with known-answer test
Date column is STRING type	DATEDIFF returns an error or null; fix CAST before this step

Assessment: Evaluated in Practical Exercise Task 3. Any pipeline using dates that produces incorrect output will fail.

BLOCK 10 — End-to-End Pipeline Practice

Hours: 1.75 | **Method:** Lab | **Time:** 1515–1700

Purpose: Trainees apply Blocks 7–9 in a complete pipeline, from raw input to typed, filtered, named output. This practice run identifies gaps before Day 3's more complex pipeline work.

TLO: The trainee will independently build a complete clean-and-transform pipeline from raw input to a typed, named, filtered output dataset — without instructor assistance.

Key Delivery Notes: - New dataset provided (not the one from earlier in the day). Trainee builds the full pipeline end to end. - Emphasize: run after each step. Check row counts. Verify output schema matches expectation.

Assessment: Not directly evaluated in Practical Exercise (the PE uses a different dataset), but directly builds the skill.

DAY 3 — PIPELINE BUILDER: JOINS; ONTOLOGY MANAGER

BLOCK 11 — Pipeline: Join Step

Hours: 2.0 | **Method:** Lab | **Time:** 0830–1030

Purpose: Joins are the most common multi-source operation and the most common source of silent data errors. Trainees must understand join types and fan-out behavior before using joins in production.

TLO: Given two datasets with a shared key, the trainee will build an inner join and a left join pipeline, identify the row count difference, handle fan-out from a many-to-one join, and produce a correct output.

Key Delivery Notes: - Join types: inner join (only rows where key exists in both), left join (all rows from left; nulls for unmatched right-side columns). Show row counts for both on the training datasets. - Key selection: if the key column names differ, rename one before joining. - Fan-out: if the right-side dataset has multiple rows per key, the join multiplies rows on the left. Always check row counts after a join. - Column selection post-join: only keep columns you need. Trim the output.

Student Activity: Join equipment dataset to a provided unit lookup table on `unit_id`. Check row count before and after. Identify whether fan-out occurred and correct it if so.

Common Errors:

Error	Detail
Fan-out ignored	"It worked" when the row count tripled — silent fan-out
Cross-project dataset reference	Both datasets must be accessible from the same Foundry project

Assessment: Evaluated in Practical Exercise Task 3. The join step is explicitly tested.

BLOCK 12 — Pipeline: Group-By, Union, Output Mode

Hours: 1.25 | **Method:** Lab | **Time:** 1045–1200

Purpose: Aggregations and union operations complete the pipeline toolkit. Output mode configuration (Overwrite vs. Append) is a governance decision that must be made intentionally.

Key Delivery Notes: - Group-By: `GROUP BY unit_id`, aggregate `COUNT(equipment_id) AS unit_equipment_count`. Only columns in GROUP BY or aggregation functions appear in output. - Union: combines two datasets with compatible schema. Handle schema mismatches with a renamed/CAST step before the union. - Output mode: **Overwrite** — pipeline replaces the output dataset on every run (point-in-time snapshot). **Append** — pipeline adds rows on every run (historical record). Most pipelines use Overwrite. Choosing wrong mode is a data governance error.

Assessment: Output mode is tested in Practical Exercise Task 3 (snapshot pipeline in Append mode, run twice, verify two records).

BLOCK 13 — Ontology Manager: Create Object Type

Hours: 2.0 | **Method:** Lab | **Time:** 1300–1500

Purpose: The Ontology is the semantic layer — the pivot point between raw data and MSS applications. Object Types must be correctly designed before being built. Property type decisions are permanent.

TLO: Given a described data entity, the trainee will create a Foundry Object Type with correctly typed properties, a Primary Key, and a display name expression — to standard.

Key Delivery Notes: - CRITICAL: Property types are immutable after objects are created. If you define `c_rating` as STRING when it should be an ENUM or INTEGER, you must delete the Object Type and all data, then recreate. Read the property type table in TM-20 Ch 4 before this block. - Property types: STRING, INTEGER, LONG, DOUBLE, BOOLEAN, DATE, TIMESTAMP. Pick the most restrictive correct type — don't default everything to STRING. - Primary Key: must be unique per Object, cannot be null. Choose carefully. - Display name expression: what humans see in Quiver or Workshop. Usually a concatenation: `[unit] - [equipment_id]`.

Student Activity: Create an `Equipment` Object Type with at least 5 correctly typed properties, a Primary Key, and a display name expression. Document property types in writing before creating them in the tool.

Common Errors:

Error	Detail
Not documenting property types before building	Leads to type correction rework
Using STRING for everything	Creates query performance issues at scale; violates SL 2 standard

Assessment: Evaluated in Practical Exercise Task 4.

BLOCK 14 — Ontology Manager: Create Link Type

Hours: 0.75 | **Method:** Lab | **Time:** 1515–1600

Purpose: Link Types model relationships between Object Types — Equipment belongs to Unit. The Link enables navigation between Objects in Quiver, Workshop, and OSDK.

TLO: Given two existing Object Types, the trainee will create a Link Type with correct cardinality and directionality — to standard.

Key Delivery Notes: - Link Type connects two Object Types. Direction matters: `Equipment` LINKS TO `Unit`. - Cardinality: MANY_TO_ONE, ONE_TO_ONE, ONE_TO_MANY, MANY_TO_MANY. Determine from the data model, not from the tool. - Cardinality is not automatically enforced by Foundry in all cases — you are declaring the intended relationship.

Student Activity: Create a Link Type from `Equipment` to `Unit`. Verify the Link appears in the Object Type view.

Assessment: Evaluated in Practical Exercise Task 5.

BLOCK 15 — Pipeline: Ontology Write Step

Hours: 1.0 | **Method:** Lab | **Time:** 1600–1700

Purpose: The Ontology write step bridges pipeline output to the Object layer. Until this step, the pipeline produces datasets — after this step, the pipeline creates or updates Objects.

TLO: Given an existing Object Type and a pipeline output dataset, the trainee will configure an Ontology write step with correct property mapping, run the pipeline, and verify Objects are created.

Key Delivery Notes: - Ontology write step configuration: target Object Type, property mapping (pipeline column → Object property), Primary Key mapping. - Property type must match: if the Object property is DATE and the pipeline column is STRING, the write will fail or produce null values. - Run the pipeline. Navigate to the Object Type in Quiver. Verify object count matches expected row count. The object count in Quiver is the ground truth — not the pipeline "ran without error."

Student Activity: Configure the Ontology write step mapping pipeline output columns to `Equipment` Object Type properties. Run pipeline. Confirm object count in Quiver matches source row count.

Common Errors:

Error	Detail
Primary Key column not mapped	All objects will overwrite each other; you end up with 1 object regardless of row count
Type mismatch between pipeline output and Object property	Silent nulls; check by opening an Object and confirming properties populated

Assessment: Evaluated in Practical Exercise Task 6.

BLOCK 15A — Object Views and Data Validation

Field	Value
Lesson Title	Object Views and Data Validation
Hours	1.5
Method	Lab
References	SL 2, Chapter 4, Tasks 4-3 and 4-4

Objective: Configure an Object View for an Object Type and use Object Explorer to validate data completeness and accuracy.

Instructor Notes: - Demo Object View configuration first, then let trainees build their own - Object Explorer validation should be taught as a standard quality gate before promotion - Emphasize that validation at this stage prevents downstream Workshop errors

---

DAY 4 — ACTIONS AND WORKSHOP APPLICATIONS

BLOCK 16 — Actions: Create Basic Action

Hours: 1.5 | **Method:** Lab | **Time:** 0830–1000

Purpose: Actions are the write-back mechanism — how authorized users update Object properties from Workshop applications. Every SL 2 data product needs at least one Action.

TLO: Given an Object Type, the trainee will create an Action with a named parameter, a write rule targeting an Object property, and an access restriction — test the Action from Ontology Manager.

Key Delivery Notes: - Action configuration: Name, Description, Parameters (name + type), Write Rules (parameter → Object property mapping), Access Control (who can execute it). - Access control: restrict to Editor role unless there is a specific reason for Viewer access. - Test from Ontology Manager (not Workshop yet). Open an Object, find the Action, execute it with a test parameter, verify the property updated. - Common mistake: no confirmation prompt configured — users can accidentally fire the Action without a review step.

Student Activity: Create an `UpdateCRating` Action with a parameter `new_c_rating` (STRING), a write rule to `Equipment.c_rating`, and Editor-only access restriction. Test from Ontology Manager.

Assessment: Evaluated in Practical Exercise Task 7. Evaluated items: parameter exists, write rule works, access restriction configured correctly (Viewer cannot execute).

BLOCK 17 — Workshop Orientation: Canvas and Table Widget

Hours: 1.75 | **Method:** Lab | **Time:** 1015–1200

Purpose: Workshop is the application layer — the interface users see and interact with.

TLO: Given a Foundry project with an Object Type, the trainee will create a Workshop application, add a table widget bound to the Object Type, and configure it to display at least 4 properties.

Key Delivery Notes: - Create application: New → Workshop Application. Name it following C2DAO naming convention. - Canvas: drag-and-drop widget placement. The canvas is freeform. - Table widget: add from widget library. Bind to Object Type — this connects it to live Ontology data. Select properties to

display as columns. - As soon as the table is bound to the Object Type, it shows live Objects. Any Action executed against these Objects will immediately update the table.

Student Activity: Create a Workshop application named per C2DAO convention. Add a table widget bound to the `Equipment` Object Type. Display `equipment_id`, `unit`, `c_rating`, `last_updated`.

Assessment: Foundation for Practical Exercise Task 8.

BLOCK 18 — Workshop: Filter, Metric, and Bar Chart Widgets

Hours: 2.0 | **Method:** Lab | **Time:** 1300–1500

Purpose: A table alone is not a dashboard. Filter, metric, and chart widgets make the application operationally useful.

TLO: The trainee will add a filter widget, metric widget, and bar chart widget to a Workshop application — each correctly bound to the Object Type data source.

Key Delivery Notes: - Filter widget: add a filter on `unit` (dropdown filter). The filter must be connected to the table widget — verify that selecting a filter option narrows the table. - Metric widget: count of Equipment Objects where `c_rating` = "C1". Bound to Object Type with a filter condition. - Bar chart widget: X axis = `unit`, Y axis = count of Objects. - Layout discipline: arrange widgets logically — filter at top, metrics below, table below that.

Student Activity: Add all three widget types. Verify filter narrows the table. Verify metric updates when Objects change. Verify bar chart displays correct unit distribution.

Common Errors:

Error	Detail
Filter widget not connected to table	Visual filter appears but table doesn't respond; check widget connections
Metric widget counting all Objects instead of filtered Objects	Connection missing

Assessment: Evaluated in Practical Exercise Task 8.

BLOCK 19 — Workshop: Connecting an Action Button

Hours: 1.25 | **Method:** Lab | **Time:** 1515–1630

Purpose: Action buttons close the loop — users can view data and update it without leaving the application.

TLO: The trainee will add an Action button to a Workshop application, trigger the Action from a selected table row, verify execution confirmation, and confirm the table refreshes to show the updated value.

Key Delivery Notes: - Add a Button widget. Configure: Action = `UpdateCRating`, Object source = selected row in the Equipment table. - The button should only activate when a row is selected. - After execution: the table should auto-refresh to show the updated `c_rating`. - Test with a Viewer-role account: the button should be disabled or absent for a Viewer.

Student Activity: Add the Action button. Select a row, click the button, confirm execution, verify the `c_rating` updated. Then switch to a test Viewer account and confirm the button is unavailable.

Assessment: Evaluated in Practical Exercise Task 9. Hard No-Go: Viewer account can execute the Action.

BLOCK 20 — Access Control: Viewer vs. Editor

Hours: 0.5 | **Method:** Discussion | **Time:** 1630–1700

Purpose: Access control is a governance requirement, not an option.

Key Delivery Notes: - Viewer: can view data, navigate applications, export if export is not restricted. Cannot edit, create, or execute Editor-only Actions. - Editor: can modify data products, execute all Actions. Should be limited to personnel who need write capability. - Principle: least privilege. Grant Viewer unless there is a specific requirement for Editor.

Assessment: Evaluated in Practical Exercise Task 10. Evaluator tests Viewer-role account.

BLOCK 20A — Builder Analysis: Contour and Quiver

Field	Value
Lesson Title	Builder-Level Analysis in Contour and Quiver
Hours	2.0
Method	Lab
References	SL 2, Chapter 6, Tasks 6-1 through 6-4

Objective: Build a saved Contour analysis with pivot table, and configure a Quiver dashboard with Object Type views.

Instructor Notes: - Distinguish from SL 1 consumer use of Contour/Quiver — builders create saved analyses and dashboards, not just one-off lookups - Saved Contour analyses become inputs to Workshop applications - Quiver dashboards are used for Ontology validation and operational data exploration

---

DAY 5 — PUBLISHING, GOVERNANCE, AND PRACTICAL EXERCISE

BLOCK 21 — Workshop Publishing and Viewer Access

Hours: 1.0 | **Method:** Lab | **Time:** 0800–0900

Purpose: An application that only you can see is not a data product.

TLO: The trainee will configure Workshop application visibility, grant a Viewer-role test account access, and confirm the test account can view the application but cannot modify data.

Key Delivery Notes: - Application visibility: set from Workshop application settings → Visibility. - Grant the test Viewer account access. Confirm access via logging in as the test account. - Verification: test account sees the application and data, but cannot execute the Action button.

Assessment: Evaluated in Practical Exercise Task 10.

BLOCK 22 — Branching

Hours: 1.0 | **Method:** Lab | **Time:** 0900–1000

Purpose: All production changes must be made on a branch, not on main. This is non-negotiable in C2DAO governance.

TLO: The trainee will create a Foundry branch, make a change on the branch, verify the change exists on the branch but not on main.

Key Delivery Notes: - Branch from within the Foundry project: Branch Manager → New Branch. Name the branch following C2DAO naming: `[unit]_[change-description]_[date]`. - Make a visible change on the branch. Verify: switch between branch and main — the change should appear and disappear. - Most common error: making the change BEFORE creating the branch. Once a change is on main, you cannot retroactively branch around it.

Assessment: Evaluated in Practical Exercise Task 11. Branch must exist with the change visible.

BLOCK 23 — Promotion Workflow

Hours: 0.75 | **Method:** Lab | **Time:** 1015–1100

Purpose: Branch changes are promoted to production through the C2DAO data steward review process. Submitting a complete, well-described promotion request is a professional skill.

TLO: The trainee will submit a promotion request from a branch with a complete change description, simulate a steward rejection, and respond to the rejection comment.

Key Delivery Notes: - Promotion request: Branch Manager → Submit for Promotion. The description field is required. The evaluator (as data steward) will reject any submission with an empty or generic description. - What a good description includes: what changed, why, and downstream impact.

Quality	Example
BAD	"Updated application"
GOOD	"Added unit filter to Workshop application per S3 request (12MAR26); filters Equipment table by unit; no impact to Ontology or pipeline"

- Rejection workflow: the evaluator responds with a comment requiring clarification. Trainees must respond and resubmit or revise.

Assessment: Evaluated in Practical Exercise Task 11.

BLOCK 24 — Full-Stack Review

Hours: 1.0 | **Method:** Review | **Time:** 1100–1200

Purpose: Before the evaluation, trainees trace their own data product end-to-end to identify gaps. This is self-assessment, not more instruction.

Key Delivery Notes: - Have trainees walk through their own product: raw file → ingestion → pipeline steps → Object Type → Link Type → Action → Workshop application → access control → branch. - Checklist: does everything connect? Does the pipeline produce the correct output? Does the Quiver Object count match source row count? Does the Viewer test account have correct access? - Instructor circulates and answers system questions — not task completion guidance.

BLOCK 25 — Practical Exercise (Evaluated)

Hours: 4.0 | **Method:** Evaluation | **Time:** 1300–1700

Scenario: S4 officer needs an equipment readiness tracker. Two provided files: equipment spreadsheet and unit location CSV. Build the full stack from scratch.

Tasks evaluated:

Task	Description
1	Create Foundry project with correct naming, markings, folder structure
2	Ingest both files; confirm row counts
3	Build pipeline: validate, clean, cast, join on `unit_id`; output clean equipment-with-location dataset

Task	Description
4	Create `Equipment` Object Type with 5 correctly typed properties
5	Create `Unit` Object Type and Link Type from Equipment to Unit
6	Configure pipeline Ontology write step populating Equipment Objects
7	Configure `UpdateCRating` Action with parameter, write rule, Editor-only access
8	Build Workshop application: table, unit/c_rating filter, C1 count metric, equipment-by-C-rating bar chart
9	Connect Action button; execute; verify table refreshes
10	Grant Viewer test account access; confirm Viewer cannot trigger Action
11	Create branch; change application header; submit promotion with complete description

Go standard: All 11 tasks without instructor assistance. Hard No-Go: Viewer account can trigger Action.

COURSE COMPLETION — NEXT STEPS

Upon receiving a SL 2 Go result, trainees have two continuation paths depending on their role:

All SL 4 tracks (WFF and Specialist) require SL 3 as a hard prerequisite.

SL 2 completion qualifies a trainee to enroll in SL 3 (Advanced Builder). Upon completing SL 3, they are eligible for all SL 4 tracks:

WFF Functional Tracks (SL 4A through SL 4F) — via SL 3: - Prerequisite chain: SL 1 → SL 2 → SL 3 → SL 4 WFF track - SL 4A: Intelligence WFF | SL 4B: Fires WFF | SL 4C: Movement & Maneuver WFF - SL 4D: Sustainment WFF | SL 4E: Protection WFF | SL 4F: Mission Command WFF - Duration: 3 days each. Role-specific MSS application for functional staff. - Who should pursue: INT/FIRES/M2/SUST/PROT/MC functional staff designated to use MSS for WFF products.

Specialist Tracks (SL 4G through SL 4O) — via SL 3: - Prerequisite chain: SL 1 → SL 2 → SL 3 → SL 4 specialist track - SL 4G: ORSA | SL 4H: AI Engineer | SL 4M: ML Engineer - SL 4J: Program Manager | SL 4K: Knowledge Manager | SL 4L: Software Engineer - SL 4N: UI/UX Designer | SL 4O: Platform Engineer - Duration: 3–5 days depending on track. - Who should pursue: 17/25-series, S6/G6, G2, data leads, and technical specialists.

T3-F (MSC Force Multiplier) — parallel path: - Prereq: SL 2 (no SL 3 required) - Duration: 5 days. Trains MSC-level data champions to advocate for and coordinate MSS adoption at battalion/brigade level. - Who should pursue: Field-grade officers, senior NCOs, and MSC staff designated as unit data leads.

Instructors should direct all trainees toward SL 3 as the next step. For trainees designated as MSC-level data champions, also highlight T3-F as an available parallel path. Discuss which SL 4 track is appropriate for each trainee and refer them to the Unit Training NCO/Officer for enrollment coordination.

USAREUR-AF Operational Data Team SL 2 Lesson Plan Outlines | Version 1.0 | March 2026