COURSE SYLLABUS

SL 4L

USAREUR-AF

COURSE SYLLABUS — SL 4L: SOFTWARE ENGINEER

Maven Smart System (MSS) — USAREUR-AF

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

DRAFT — NOT FOR OFFICIAL USE. FOR TRAINING PLANNING PURPOSES ONLY.

26 MARCH 2026

DRAFT — UNOFFICIAL — NOT FOR OPERATIONAL USE

COURSE SYLLABUS — SL 4L: SOFTWARE ENGINEER

MAVEN SMART SYSTEM (MSS) — USAREUR-AF

Field	Detail
Level	SL 4L — Software Engineer Specialist Track
Duration	5 days (40 hours)
Prerequisites	SL 1, SL 2, SL 3 (all Go evaluations on file — **REQUIRED**, not recommended); working TypeScript or Python proficiency; familiarity with REST APIs and async patterns
Audience	Software engineers building external applications and platform integrations on MSS
Format	Instructor-led lab + guided practice + practical exercise
Location	MSS Training Environment (OSDK developer access required; external IDE permitted)

BLUF: SL 4L teaches software engineers to build production-quality applications on MSS — reading and writing the Foundry Ontology via OSDK, manipulating datasets via the Platform SDK, implementing computed properties as TypeScript Functions on Objects, writing Action validators with business logic, and delivering Slate applications integrated with the Foundry API. SL 4L is for developers who need to go beyond the MSS UI. If Workshop and Pipeline Builder cover the requirement, SL 3 is sufficient.

LEARNING OBJECTIVES

#	Objective
1	Authenticate to the Foundry Ontology via OSDK, execute a paginated filtered object query using `ResourceIterator`, and handle multi-page result sets
2	Traverse Link Types via OSDK to query related objects across Object Type relationships
3	Execute an Action via OSDK with async handling, polling for completion, and structured error handling
4	Subscribe to Object change notifications via OSDK WebSocket and implement bulk operation patterns

#	Objective
5	Read and write Foundry datasets using the Platform SDK with transaction management and branch operations
6	Build a TypeScript Function on Objects (FOO) implementing a computed property using bulk query patterns — no N+1 queries
7	Write and test a TypeScript Action validator with at least three distinct validation conditions, including cross-field logic
8	Build a Slate application displaying live Ontology data with Action triggers, state management on Action completion, and error state handling
9	Complete a C2DAO code review and deployment workflow including credential handling, input sanitization, and the deployment checklist

PRE-COURSE CHECKLIST

Complete **10+ duty days before Day 1:**

- ☐ Request **OSDK developer access** from C2DAO — distinct from standard MSS Builder access; requires a developer token and OSDK access for the specific Object Types used in training; unit MSS Administrators cannot provision this
- ☐ Configure your laptop: Node.js LTS, TypeScript, and your preferred IDE (VS Code recommended); bring your configured laptop — external IDE is permitted and recommended
- ☐ Read TM-40L, Chapter 1 (Introduction, role, the 5-layer data stack) — 25 min
- ☐ Read TM-40L, Chapter 9 (Security and Compliance) — read before the labs, not after; security requirements affect every code pattern

DAILY SCHEDULE

Day 1 — OSDK Fundamentals

Time	Block	Method	Content
0800–0900	1	Lecture	SWE role on MSS; the 5-layer data stack; OSDK architecture — client, token, type generation, how it differs from standard REST

Time	Block	Method	Content
0900–1100	2	Lab	OSDK setup: authentication architecture, client initialization, token handling, first object query (single page)
1100–1115	—	Break	
1115–1200	3	Lab	OSDK: filtering and sorting — query predicates, sort expressions, field selection
1200–1300	—	Lunch	
1300–1500	4	Lab	OSDK: pagination — `ResourceIterator`, iterating through all pages, handling multi-page result sets
1500–1515	—	Break	
1515–1700	5	Lab	OSDK: Link traversal — querying related objects across Link Types; join-like patterns without leaving the Ontology

Evening reading: TM-40L, Chapter 2 (OSDK Fundamentals) — full chapter; the lab moves fast.

Day 2 — OSDK Advanced

Time	Block	Method	Content
0800–0830	—	Review	Day 1 questions; pagination checkpoint — confirm all trainees can iterate through a multi-page result set
0830–1030	6	Lab	OSDK: Action execution — submitting Actions via OSDK, async response pattern, task ID polling for completion
1030–1045	—	Break	
1045–1200	7	Lab	OSDK: error handling and retry — Action execution failures, timeout handling, structured error response patterns
1200–1300	—	Lunch	
1300–1500	8	Lab	OSDK: Object subscriptions — real-time change notifications via WebSocket; connecting subscriptions to application state
1500–1515	—	Break	

Time	Block	Method	Content
1515–1700	9	Lab	OSDK: bulk operations — batch queries, bulk Action submissions; avoid per-object API calls in loops

Evening reading: TM-40L, Chapter 3 (OSDK Advanced) — subscriptions and bulk operations; Chapter 4 (Platform SDK) — transaction patterns and error handling sections.

Day 3 — Platform SDK and TypeScript Functions on Objects

Time	Block	Method	Content
0800–0830	—	Review	Day 2 questions; async Action execution pattern review — confirm understanding of task ID polling
0830–1030	10	Lab	Foundry Platform SDK: dataset read operations, write transactions, file resources, branch management
1030–1045	—	Break	
1045–1200	11	Lab	Platform SDK exercise: build a dataset integration using Platform SDK read/write transaction pattern
1200–1300	—	Lunch	
1300–1500	12	Lab	TypeScript Functions on Objects (FOO): repository structure, computed property implementation, function registration
1500–1515	—	Break	
1515–1700	13	Lab	FOO: bulk query patterns — avoiding N+1 queries; testing with a 200+ Object set; performance validation

Evening reading: TM-40L, Chapter 5 (TypeScript FOO) — testing patterns and N+1 avoidance; Chapter 6 (Actions with Complex Validation) — review validator architecture before Day 4.

Day 4 — Action Validators and Slate Applications

Time	Block	Method	Content
0800–0830	—	Review	Day 3 questions; FOO performance patterns debrief

Time	Block	Method	Content
0830–1030	14	Lab	TypeScript Action validators: multi-condition validation, cross-field logic (`if status = DEPLOYED, location must be non-null`), error message standards
1030–1045	—	Break	
1045–1200	15	Lab	Validator testing: writing a test suite — 4 valid inputs, 4 invalid inputs; each test case paired with its expected error message
1200–1300	—	Lunch	
1300–1500	16	Lab	Slate applications: application structure, Foundry API integration, widget binding, initial data load
1500–1515	—	Break	
1515–1700	17	Lab	Slate: state management on Action completion — trigger state variable on Action completion (not a timer); error state display for failed Actions and empty dataset responses

Evening reading: TM-40L, Chapter 7 (Slate) — state management patterns; Chapter 8 (CI/CD and Code Repository Discipline) — read before Day 5.

Day 5 — CI/CD, Security, and Practical Exercise

Time	Block	Method	Content
0800–0900	18	Lecture	CI/CD: code repository discipline, PR workflow, automated testing integration, C2DAO deployment checklist
0900–1000	19	Lecture	Security and compliance: OSDK token handling (no hardcoded tokens — ever), input sanitization, OPSEC for application code, common Foundry integration vulnerabilities
1000–1015	—	Break	
1015–1100	20	Brief	Practical exercise scenario brief; planning time
1100–1200	—	Buffer	Questions / environment check

Time	Block	Method	Content
1200–1300	—	Lunch	
1300–1700	21	**Eval**	**Practical exercise:** OSDK query → Action validator → Slate UI → deployment checklist, end-to-end

REQUIRED READING

When	Reading
Before Day 1	TM-40L, Ch 1 (Introduction, 5-layer data stack)
Before Day 1	TM-40L, Ch 9 (Security and Compliance) — read first
Day 1 evening	TM-40L, Ch 2 (OSDK Fundamentals) — full chapter
Day 2 evening	TM-40L, Ch 3 (OSDK Advanced — subscriptions/bulk)
Day 2 evening	TM-40L, Ch 4 (Platform SDK — transactions/error handling)
Day 3 evening	TM-40L, Ch 5 (FOO — testing/N+1 avoidance)
Day 3 evening	TM-40L, Ch 6 (Actions with Complex Validation)
Day 4 evening	TM-40L, Ch 7 (Slate — state management)
Day 4 evening	TM-40L, Ch 8 (CI/CD and Repository Discipline)

PRACTICAL EXERCISE

Scenario: Build a unit readiness application — a Slate front-end that queries Unit Objects filtered by readiness status, allows an authorized user to update status via a validated Action, and handles errors gracefully.

#	Task
1	Authenticate via OSDK; execute a paginated query of Unit Objects filtered by `readiness_status` — use `ResourceIterator`, iterate through all pages
2	Write a TypeScript Action validator that enforces: required field presence, valid status enumeration, cross-field logic (`if status = DEPLOYED, location must be non-null`)

#	Task
3	Build a TypeScript Function on Objects that computes a `days_since_last_update` property using bulk query patterns
4	Build a Slate application: table view of Unit Objects with status-color formatting; Action trigger on row click with confirmation modal; error state display for Action failures and empty dataset responses
5	Write a test suite for the Action validator: minimum 8 test cases (4 valid, 4 invalid); all 8 must pass — the evaluator will run the test suite against your code
6	Complete the C2DAO code review checklist and deployment documentation; confirm no credentials are hardcoded anywhere in the application

Go standard: Pass 5 of 6 tasks. Validator passes all 8 test cases — partial credit does not apply. Slate application handles error states without crashing. Deployment checklist complete with no hardcoded credentials.

GO CRITERIA

Task	Hard Standard
Validator test suite	Evaluated by running provided test cases against your actual code — all 8 must pass; no partial credit
Slate error handling	Evaluator will intentionally trigger an error condition (Action failure, empty dataset) — crash or unhelpful error = No-Go; display a user-visible error message
Deployment checklist	No credentials in code, no tokens hardcoded, all secrets via environment variables or the Foundry Secrets API — hardcoded token anywhere fails

KEY TIPS

Risk	Guidance
Pagination	Use `ResourceIterator` and iterate through all results — every time; evaluator uses an Object set requiring more than one page
Action execution	OSDK Action execution is asynchronous — you get a task ID, not a result; poll for completion or use `await` correctly; synchronous treatment will not catch failed Actions

Risk	Guidance
TypeScript FOO performance	Avoid per-object API calls in a loop; use bulk query patterns from Chapter 5; evaluator tests with a 200+ Object set
Slate state management	When the user triggers an Action, the table must refresh — set up a state variable tied to Action completion, not a timer; Chapter 7 has the pattern
Credential handling	Check every file before submitting — the answer is always no credentials in code, all secrets via environment variables or the Foundry Secrets API

CONTINUATION

Graduates designing platform architecture or establishing development standards for a software engineering team may pursue **SL 5L (Advanced Software Engineering)**. SL 5L covers OSDK-first architecture patterns, enterprise CI/CD, TypeScript advanced patterns, security review processes, developer platform toolchain design, and evaluated platform architecture builds. Prerequisites: SL 4L Go evaluation on file; 18+ months active SWE experience on MSS or equivalent Foundry/OSDK platform.

ASSOCIATED EXERCISES AND ASSESSMENTS

Item	Reference
Pre-course exam	EXAM_TM40L_PRE
Post-course exam	EXAM_TM40L_POST
Practical exercise	EX_40L (EXERCISE.md + ENVIRONMENT_SETUP.md)

RELATIONSHIP TO WFF TRACKS

WFF track analysts (SL 4A through SL 4F) are the operational consumers of applications and integrations built in this course. Software engineers should understand how each WFF community uses MSS tooling: mission command analysts (SL 4F) use Slate applications for operational dashboards; intelligence

analysts (SL 4A) rely on OSDK-backed applications for fused product displays; all WFF communities benefit from validated Action logic, reliable pipeline integrations, and Slate UIs that surface Foundry data without requiring analysts to write code.

USAREUR-AF Operational Data Team Syllabus SL 4L | Version 2.0 | March 2026