Daml Certification Syllabus

Daml Contract Developer

January 2021

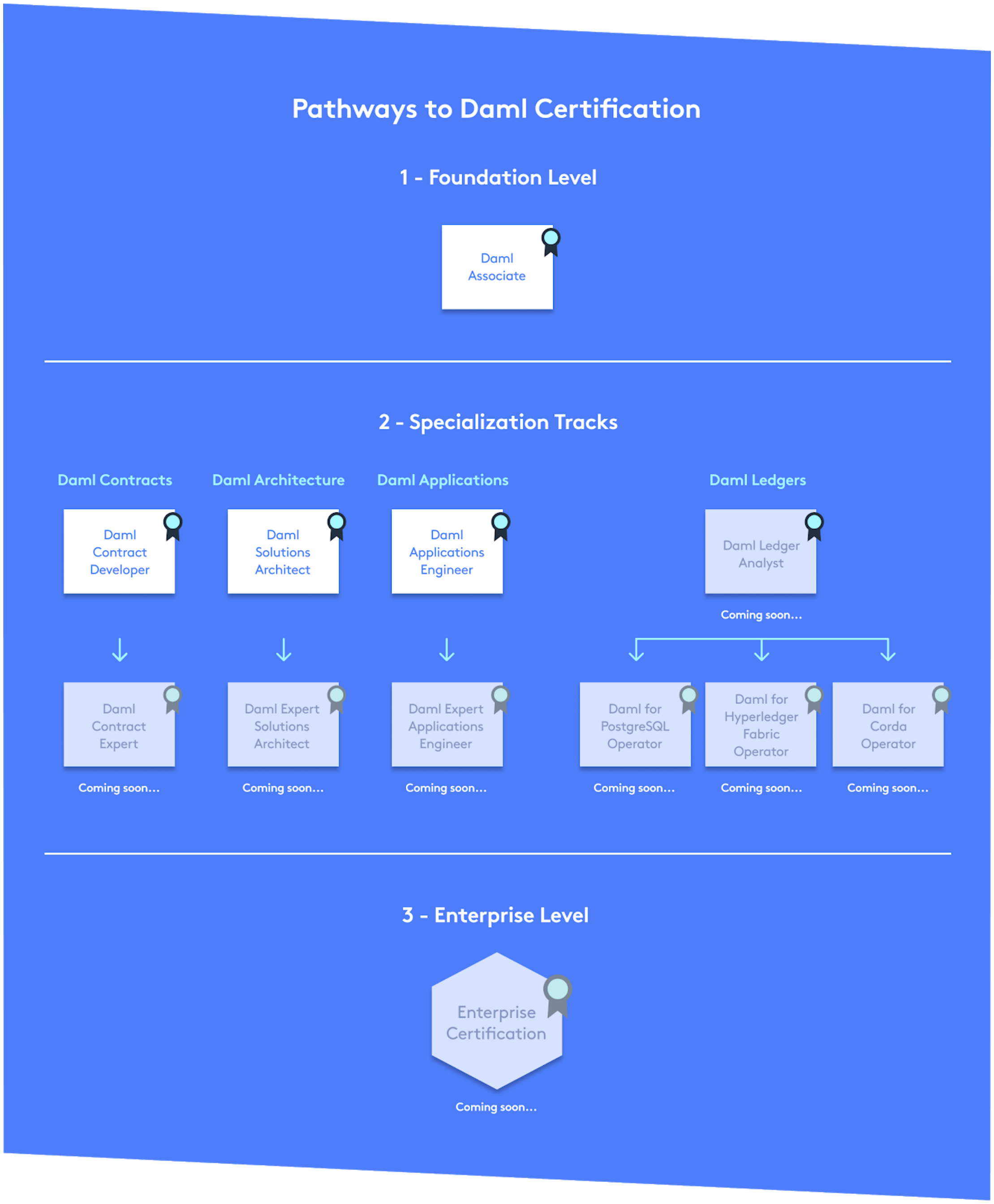
Introduction

Daml Contract Developer Exam

This first exam in the Daml Contracts Track covers the mechanics of Daml: The Daml Ledger Model, Daml's surface syntax and how it links to that model, your ability to translate between real world workflows and Daml code, and your know-how regarding the testing and debugging of Daml Code.

Daml-Certification Program

The Daml Contracts exams test a developer's skills in expressing the core of a distributed application using Daml Smart Contracts. It covers theoretical knowledge of the underlying abstractions, hands-on development know-how, testing capabilities, as well as common patterns, best practices, and the ability to design applications for performance and security.



Pathways to Daml Certification

1 - Foundation Level

Daml Associate

2 - Specialization Tracks

Daml Contracts
- Daml Contract Developer
- Daml Contract Expert — Coming soon…

Daml Architecture
- Daml Solutions Architect
- Daml Expert Solutions Architect — Coming soon…

Daml Applications
- Daml Applications Engineer
- Daml Expert Applications Engineer — Coming soon…

Daml Ledgers
- Daml Ledger Analyst — Coming soon…
- Daml for PostgreSQL Operator — Coming soon…
- Daml for Hyperledger Fabric Operator — Coming soon…
- Daml for Corda Operator — Coming soon…

3 - Enterprise Level

Enterprise Certification

Coming soon…

Prerequisites

- Daml Associate Certification
- Solid understanding of standard computer science concepts
- Installation of Daml SDK, or access to the web SDK at daml.com/learn

Exam Outline

Exam Organization

The exam takes approximately 90 minutes to complete and will be conducted online at the time of your choosing. It consists of 39 questions of different types, ranging from technical definitions to code analysis and translating requirements into code. It is expected that candidates will use the Daml developer documentation and SDK as aids during the exam.

The 39 exam questions will be separated into four main topics. The total score on each topic counts as a percentage of your final score as shown below. Each topic will be scored separately, with all questions for a topic having the same weight.

The exam score will be the total of the topic scores. To be Daml-certified, you must have a total exam score of 70%.

Exam Topics

Daml Contract Language (60%)

- Template structure, keywords, and their link to the Ledger Model
- Daml's type system for data, inbuilt types, immutability, the keywords and syntax used to define custom data types, serializability, and some basics on typeclasses
- Pattern Matching and control flow - `case` and `if` expressions
- Actions and `do` notation in Daml
- The Update Action and its link to the Daml Ledger model
- Function signatures and defining custom functions
- "Looping" - folds and recursion
- Errors, Aborts, and Assertions
- Daml Projects, Packages, Modules, Imports, and dependencies
- Basic Standard Library functions on Tuples and Lists
- Searching the Standard Library by signature and name

Testing and Debugging (17.5%)

- Functionality of tooling - Scenarios, Script, REPL, Navigator
- Differences between IDE-integration `daml test`/`daml test-script` and `daml script`
- Translation between Scenario/Script View and Ledger Model
- Testing Daml Models via API and Navigator
- The debug and trace functions

- Testing for failures

Daml Ledger Model (17.5%)

- Ledger structure in terms of Commits, Actions, Subactions, Transactions, Subtransactions, Immutability
- Relationships between parties and contracts: Signatories, Observers, Stakeholders, Maintainers, Controllers, Actors, Informees, Witnesses
- Guarantees the Daml Ledger Model gives
- Party Projections of transactions
- Action types and their authorization rules
- The time model of the ledger, and the guarantees on Ledger Time
- Immutability, contract state, and consistency of Daml ledgers

Daml Execution Model (5%)

- Daml execution model in terms of Interpretation, Validation, and Commit, and what happens during each phase
- High- level link between Daml, Daml-LF, and Transactions
- Different types of errors and where they occur during execution
- The semantics of contract and key lookups during command interpretation

Example Questions

Correct answers are marked in **bold**

Random Numbers

Ledger Time is a source of random numbers as no single party can manipulate it.

- True
- **False**

Causal Monotonicity

Ledger Time is guaranteed to satisfy causal monotonicity. What does that mean? Select the correct option.

- Archives always come after all other actions on a contract
- All actions on a contract are totally ordered by time
- Actions on a contract are ordered by when they were submitted
- Actions on a contract are ordered by when they were committed
- **Creates always precede all other actions on a contract**

Debug and Trace

Where are messages from the `debug` and `trace` functions put out? Select all that apply.

- **The bottom of the Transaction View for Scenarios/Scripts in the IDE**
- The bottom of the Table View for Scenarios/Scripts in the IDE
- **The console output of Daml REPL**
- **The log output of Daml Script**
- **The log output of Ledger Nodes**

Test Tool Architecture

Which of these tools allow you to test/debug a Daml model through the Ledger API, allowing you to perform the same action any other client application could take? Select all that apply.

- **Navigator**
- **Script**
- **REPL**
- Scripts run by the IDE / Studio

Technical Architecture

The [**Daml Compiler**, DAR, Daml-LF, Daml Engine, JVM] consumes Daml code files and emits a [Daml Compiler, **DAR**, Daml-LF, Daml Engine, JVM] containing DALF files, which are binary encoded messages from the Daml-LF schema. The PackageService consumes a DAR file and emits [Daml Compiler, DAR, **Daml-LF**, Daml Engine, JVM] packages. When a command gets submitted, the [Daml Compiler, DAR, Daml-LF, **Daml Engine**, JVM] interprets the Daml-LF in a [Daml Compiler, DAR, Daml-LF, Daml Engine, **JVM**] process to generate a transaction.

Finding out StdLib Function Signatures

What's the full signature (including typeclass bounds) of the Standard Library function `Prelude.sequence`?

Applicative m => [m a] -> m [a]