SR15 Validation Tool - POC overview

Objective	To define the scope of the POC for the build-out of a Validations Tool (VT) for the SR15 release
Due date	17 Feb. 2025
Driver(s)	Sam Nazha
Approver(s)	Farid Sammur
Informed	Craig White Desmond Hokin Jimmy Halstead
Status	APPROVED

Version history

Version	Date	Details
1.0	13 Feb. 2025	Initial version approved by FS

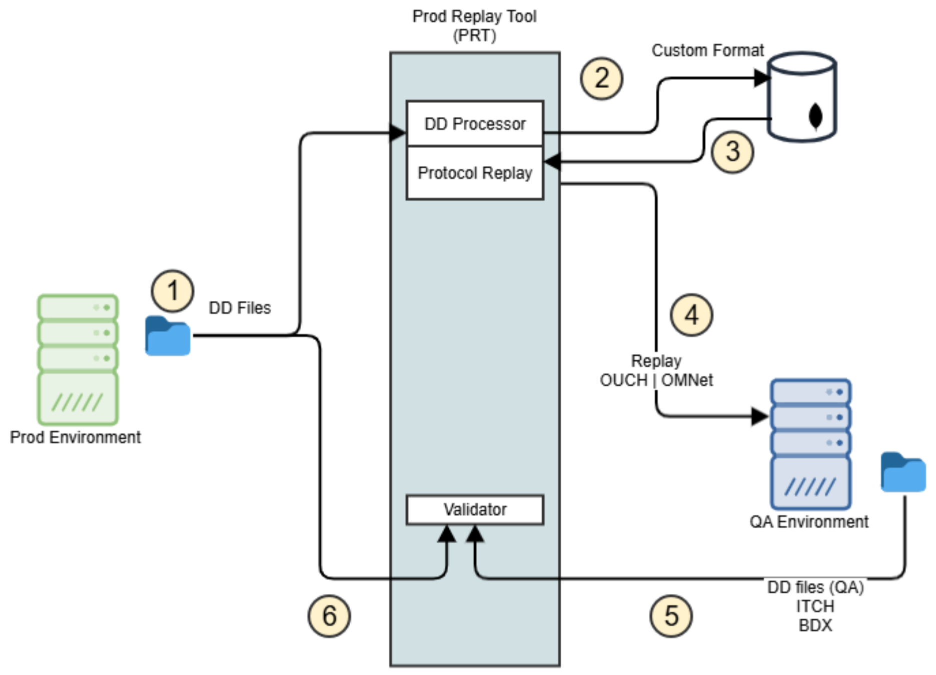
Introduction

A Production Replay Tool (PRT) has been written to allow for the reconstruction of trade messages (OUCH | OMNet protocol) from DD files; messages are then fed back into the QA environment and reconciled (at a high level) to report on differences.

This POC's purpose is to provide extended validation capabilities (at a per-*object* level) outside the existing PRT tool. In doing so we can retain the replay characteristics of the PRT, and buildi out a richer set of configurable, rule-based validations that can scale with changes in platforms and behaviours.

Current State

The following diagram highlights the flow and capabilities of the PRT.



PRT Overview

DD file processing

1. DD files are extracted from production and provided as inputs
2. The PRT converts the records in the DD file into a custom format and persists them into the database
    a. As the DD contains trades across all protocols, the format is generic to handle all

Replay functionality

When a replay operation is initiated by a user the following actions are performed

1. Data is loaded from the database
2. Data is processed to infer the source protocol that resulted in the Trade
    a. OMNet or OUCH messages are constructed
3. Protocol messages are then fed into the QA environment

Validation features

Once trades are fed through the QA environment (synchronously) the new DD files can be extracted and compared against the production files that were initially loaded.

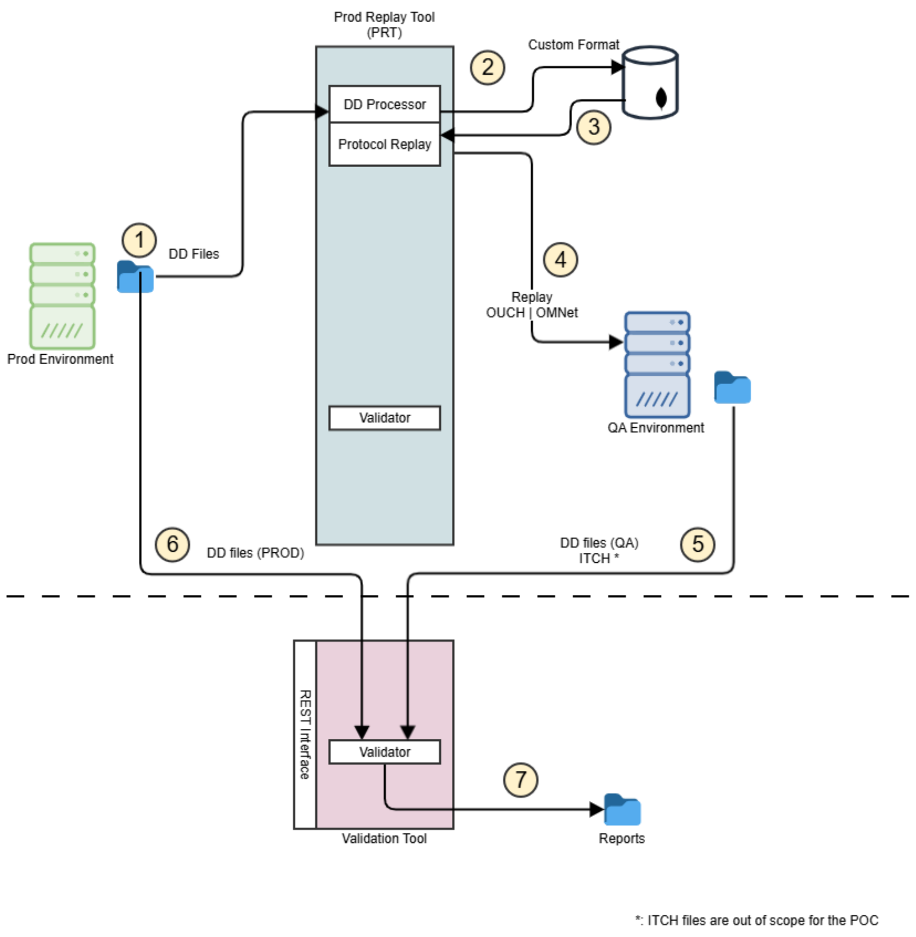
1. Load new DD files and ITCH/BDX messages from QA
2. Compare each trade message and stop at the first mismatch

Desired State

The PRT provides a reasonable facility to replay messages in the QA environment, and a rudimentary validation/check is implemented.

Given the generic validation implemented for replayed trades (from QA) against the source trade set (DD file production) we want to have a richer and more configurable path for validation. As such, this new initiative will expand the test capabilities **around** the PRT, specifically for the validation function, without making any changes to it.

The following diagram shows the scope and boundary of the Validation Tool, highlighted below the dotted line.

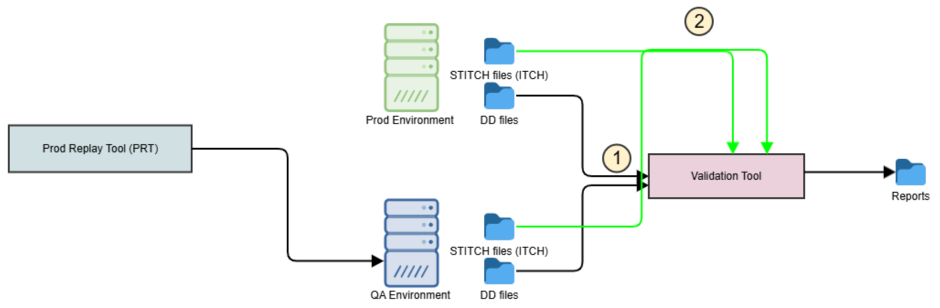


Scope

To deliver a new, streamlined, flexible and configurable testing experience for ASX Trade that can be run in an on-prem environment.

The Validation Tool (VT) will be built to provide the following capabilities across two phases, as indicated in the diagram below.

It is essential to build the tool in a manner that can be easily scaled to support additional protocols without structural or intrusive changes.



In-scope:

- Building a new Validation Tool that can handle additional message types in future phases
- Full support for validating DD files
- Basic support for validating STITCH files
    - Validation of up to 5 fields
- Producing result reports in a CSV or similar format

Out of scope:

- Full validation of all fields in ITCH messages (left for Phase 2)

The low-level design and implementation should take into consideration the addition of support for more message types in future phases of the project.

Requirements

Item	Summary	Details
R1	Compare trades and order book changes	Messages in DD files originate from a variety of protocols and can represent different actions (e.g. order, trade, clearing-trade). Comparison of messages across files can be built, and support for additional message types can be added in the future. The following message types are to be supported in phase 1. 1. Trade 2. Clearing Trade 3. Order book changes
R2	Field-level differences are captured and categorised according to rules	Categorise (message) differences into several categories • INFO: difference is observed but does not present any consequences • ACCEPTED: difference is observed but is accepted according to a specific rule (reference to rule number) • WARNING: difference observed but is not fatal; testing can of other trades / message can continue • FATAL: difference is not accepted and testing should not proceed until difference is addressed
R3	Ability to specify rules to classify differences	All rules will be driven by configuration, no built-in rules will be bundled into the application. Rules will be maintained separately, and additions/changes in the ruleset will not require any code changes or releases Example: • differences in `trade_date` are acceptable if < 3 days • differences in `trade_date` are warnings if: 3 days < dates < 10 days • differences in `trade_date` > 10 days are fatal • trade_price > original trade_price + delta is fatal if delta >= 0.1 • diffs in date_time can be ignored • diffs in amount are acceptable without a tolerance of +=0.01
R4	Support for various data types	• timestamp • long / integer • boolean • string • key/value pairs
R5	Basic support for ITCH messages	ITCH messages will be provided in STITCH files; we want basic validation of ITCH files (e.g. compare a handful of fields, classify diffs and generate reports). This is to ensure the solution can scale to any message

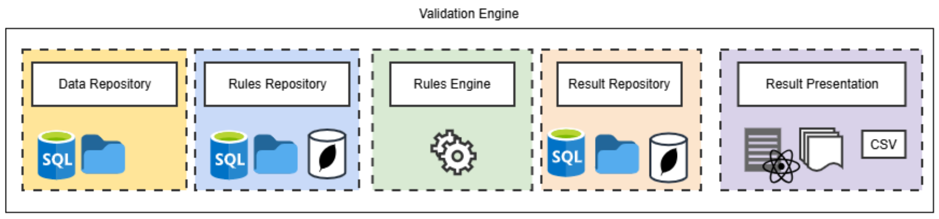
Constraints

- Access to the PRT source code will not be possible (contractual restrictions)
- No changes can be considered for the PRT
- Validations will be flexible and maintained via a configuration file that can be managed without the need for a software release
- All code will be deployed in an on-prem setting without any dependencies on Cloud components or services
- All technology choices and requirements will need to communicated beforehand

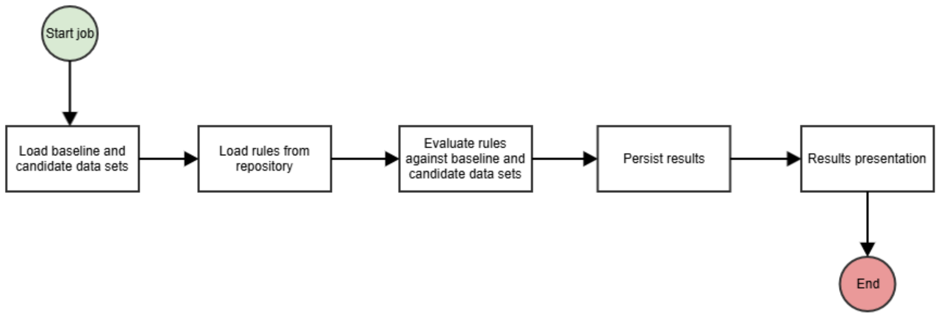
Implementation Guide

High Level Design

The goal is to build a scalable and customisable tool from day-1; the following HLD provides the basis on how this can be achieved.



All the components of the HLD are brought together via minimal code in the Validation Engine; this can easily be achieved by utilising standard design patterns, interfaces and configuration. The following flow diagram shows the expected flow within the Validation Tool.


Start job
Load baseline and candidate data sets
Load rules from repository
Evaluate rules against baseline and candidate data sets
Persist results
Results presentation
End

The following psedo-code highlights how these components are brought together in accordance with the flow diagram are shown in the following code snippet.

Pesudo code for the validation engine

```
public void run(final JobParameters jobParams) {
                val baselineData = dataRepository.load(jobParams.baseline);
                 val candidateData = dataRepository.load(jobParams.candidate);
                val rules = rulesRepository.loadRules();
                val evaluationResult = rulesEngine.evaluate(rules, baselineData, candidateData);
                resultRepository.write(evaluationResult);
                resultsPresenter.present(evaluationResult);
}
```

Terminology

All code and documentation shall use **consistent terminology** with this document, most notably:

- baseline: the reference data set to compare *against* (this is typically production data, but it does not have to be)
- candidate: the new or updated data set to compare against the baseline
- all date representations shall use an integer in the YYYYMMDD format

Components

Data Repository

DD and STITCH files will be provided in a directory, and categories according to the environment (e.g. QA, IWT, PROD).

There should be a standard interface to load data from any underlying repository, but the scope of this POC and Phase 1 will be to load data in files in directories.

The data model used by the repository will handle all protocol-specific data formats (DD, ITCH, etc).

Rules Repository

Rules can be sourced from any repository; for the scope of this POC and Phase 1 all rules will be loaded a set directory. As a result rules will be loaded in at runtime and changes in rulesets do not require a rebuild of the application.

Rules Engine

The evaluation of the rules against the data should leverage a rules engine, with rules specified in json/xml/yaml or another suitable format.

Rules should be grouped by protocol or source.

Result Repository

For the POC and Phase 1, the results of the evaluation (categorisation of differences according to rules) should be written to repository in a standardised format to allow for the presentation of that information in any structure.

Rules Presentation

The choice of presentation will be driven by configuration; for the POC and Phase 1 the results will be written to a CSV or Excel file.

Configuration

All aspects of the tool will be controlled and driven by configuration, and it is envisaged that configuration will dictate items like:

- location of inputs and outputs
- types of repositories, presenters, etc
- individual settings

Rules

Rules will be grouped by source format:

- DD files
- STITCH files

Rules will be able to classify differences into several categories:

- INFO: difference is observed but does not present any consequences
- ACCEPTED: difference is observed but is accepted according to a specific rule (reference to rule number)
- WARNING: difference observed but is not fatal; testing can of other trades / message can continue
- FATAL: difference is not accepted and testing should not proceed until difference is addressed

At the time of this writing the full ruleset is not known, and rule formation will require input from ASXTrade SMEs.

Given the nature of this collaborative exercise it will take several attempts to finalise, as a starting point the following requirements for rule classification will apply:

- any differences between fields between baseline and candidate will be classified as `WARNING`
- any missing fields will be an `FATAL` (field in baseline but not in candidate)
- any additional fields will be `INFO` (field in candidate but not in baseline)
- any missing records in baseline but not in candidate will be `FATAL`
- any additional records in candidate but not in baseline will be `FATAL`

The rules shall be stored in a format conducive to smooth collaboration with ASXs (e.g. yaml, json, or any other reasonable representation).

Multiple files may exist per protocol; there are no limits.

For guidance, a sample representation of a rule is presented below.

dd-rules-1.json

```
{
    "rules": [
        {
            "name": "Contract size change rule",
            "label": "DD-OrderBook-ContractSize-1",
            "check": "$baseline.contractSize() != $candidateSize.contractSize()",
            "classification": "WARNING",
                        "description": "Contract sizes must be equal"
        },
        {
            "name": "Contract size missing in candidate",
            "label": "DD-OrderBook-ContractSize-2",
            "check": "$baseline.contractSize() != null && $candidateSize.contractSize() == null",
            "classification": "FATAL",
                        "description": "Contract size is mandatory",
        },
        {
            "name": "Contract size missing in baseline",
            "label": "DD-OrderBook-ContractSize-3",
            "check": "$baseline.contractSize() == null && $candidateSize.contractSize() != null",
            "classification": "INFO",
                        "description": "Contract size detected"
        },
        {
            "name": "Strike price diff is above threshold",
            "label": "DD-OrderBook-StrikePrice-1",
            "check": "Math.abs($baseline.strikePrice() - $candidateSize.contractSize()) >= 0.01",
            "classification": "ERROR"
        },
        {
            "name": "Currency diffs are fatal",
            "label": "DD-OrderBook-Currency-1",
            "check": "$baseline.currency() == $candidateSize.currency())",
            "classification": "FATAL"
        }
    ]
}
```

Directory Layout

The POC shall read DD files from a specific directory and write (CSV) reports to a specific directory as well.

Both these directories will be set in the application's configuration.

Sample configuration

```
input.dd.dir=/opt/validation_tool/input
output.result.dir=/opt/validation_tool/output
```

The input directory will contain the DD files for the baseline and comparison data set, and will conform to the following format:

`<input.dd.dir>/<ENV>/<DD-file-date>/<DD-files>`

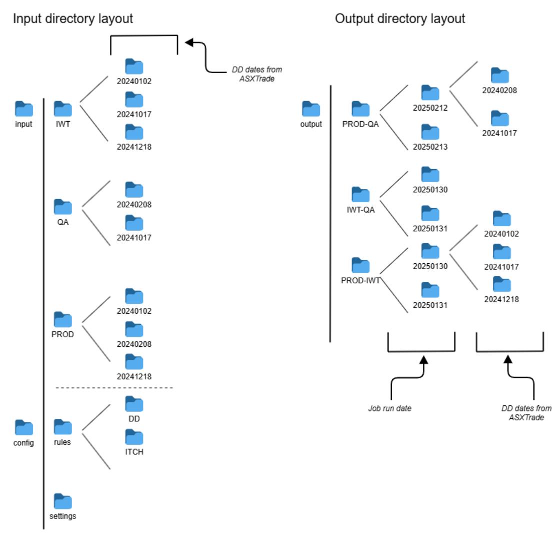
For example, the `<input.dd.dir>/PROD/20250125/<DD-files>` directory would contain the production DD files for 27-Jan-25.

Similarly, the output directory will be used to write diff results and corresponding CSV reports to, and will conform to the following format:

`<output.result.dir>/<BASELINE_ENV>-<CANDIDATE-ENV>/<DD-file-date>/<rundate><result-files>`

For example, the `<output.report.dir>/PROD-IWT/20250127/20250213/<result-files>` directory would contain the comparison result for DD files for 27-Jan-2025 when IWT (candidate) DD files (baseline) are compared against PROD (baseline) DD files for the job run on 13/02.

The following diagram summarises both directory structures.



Results

Difference results are generated from execution of the rules according to the following pseudo-code:

```
val rulesEngine = new RulesEngine(config.rulesFiles);
val diffResult = rulesEngine.evaluate(baseline, candidate);
resultRepository.write(diffResult);
```

For the POC the results are to be written to a directory in JSON format.

The format of the file name will be:

`<output.result.dir>/<BASELINE_ENV>-<CANDIDATE-ENV>/<DD-file-date>/<rundate>/<job-id>.json`

For example, for a job run on 13/02 for DD files generated on 27/01 the target would be: `<output.report.dir>/PROD-IWT/20250127/20250213/2fe3169c-0fac-4aee-a278-d4ba7ac3d55d.json`

The JSON format for the results could look like the following fragment:

Sample JSON result

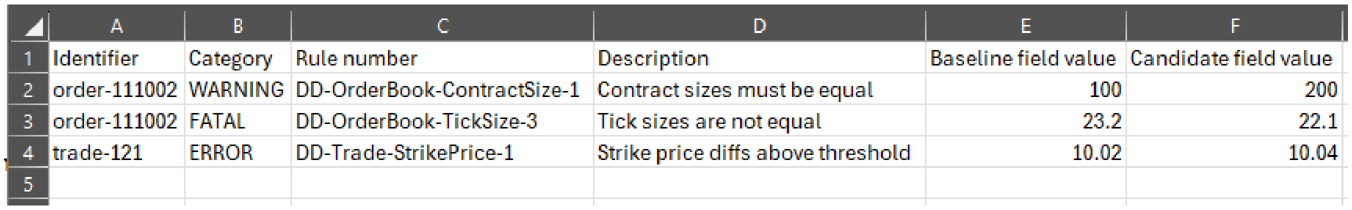
```json
{
    "rundate": "13/02/2025T14:02:23.123Z",
    "baseline": {
        "env": "PROD",
        "label": 20250123
    },
    "candidate": {
        "env": "QA",
        "label": 20250124
    },
    "summary": {
        "status": "Differences detected",
        "count": {
            "INFO": 3,
            "ACCEPTED": 12,
            "ERROR": 17,
            "FATAL": 2
        }
    },
    "differences": [
        {
            "id": "order-111002",
            "category": "WARNING",
            "rule_number": "DD-OrderBook-ContractSize-1",
            "description": "Contract sizes must be equal",
            "values": {
                "baseline": 100,
                "candidate": 200
            }
        },
        {
            "id": "order-111002",
            "category": "FATAL",
            "rule_number": "DD-OrderBook-TickSize-3",
            "description": "Tick sizes are not equal",
            "values": {
                "baseline": 3,
                "candidate": 4
            }
        },
        {
            "id": "trade-121",
            "category": "ERROR",
            "rule_number": "DD-Trade-StrikePrice-1",
            "description": "Strike price diffs above threshold",
            "values": {
                "baseline": 10.02,
                "candidate": 10.04
            }
        }
    ]
}
```

Report Presentation

Reports are generated from results **after** they are written to the result repository.

```
val reportPresenter = new ReportPresentor(jobParams.reportType);
val rulesEngine = new RulesEngine(config.rulesFiles);
val diffResult = rulesEngine.evaluate(baseline, candidate);
resultRepository.write(diffResult);
reportPresentor.generate(diffResult)
```

For the POC the report to be generated is a CSV file; however the implementation should be flexible to "present" different report types without changing the core flow of the engine.



	A	B	C	D	E	F
1	Identifier	Category	Rule number	Description	Baseline field value	Candidate field value
2	order-111002	WARNING	DD-OrderBook-ContractSize-1	Contract sizes must be equal	100	200
3	order-111002	FATAL	DD-OrderBook-TickSize-3	Tick sizes are not equal	23.2	22.1
4	trade-121	ERROR	DD-Trade-StrikePrice-1	Strike price diffs above threshold	10.02	10.04
5						

Operational Details

On tool startup the application ....

- will load the relevant configuration from the file system and configure itself to use the relevant components (e.g. SQL rules repo, CSV results presenter, etc)
- will scan the <input> directory and be ready to service validation requests on subdirectories
  - e.g. <input>/QA, <input>/PROD, <input>/IWT, etc

On job start request received ...

- The VT will run for datasets referenced in the job parameters
  - It should be permitted to compare any combination of environments (e.g. QA vs IWT, QA vs PROD, etc)
  - Labels will be used to identify data sets within each environment
- A job will be started and a unique job ID returned to the requester

At any time requesters can ...

- query the job status by job ID
- obtain results by job ID
  - A path to the results file can be provided for download

Interfaces

Both CLI and API interfaces can be used interchangeably and provide identical capabilities, thus allowing end-users to use the VT in addition to bespoke integrations in the future.

API

Interaction with the tool will be provided via RESTful APIs and command-line interaction, and both will provide the same functionality and can be used interchangeably.

The following is the swagger spec for the API operations.

OpenAPI spec

```
openapi: 3.1.0
info:
  title: Validation Tool
  description: |-
    This OpenAPI 3.1 specification defines the interface for interacting and using the Validation Tool. You can
find out more at [this page](https://confluence.asx.com.au/display/~nazha_s/SR15+Validation+Tool+-
+POC+overview).

  contact:
    email: sam.nazha@asx.com.au
  version: '0.1'
tags:
  - name: Environment
    description: Input files for each environment
  - name: Job
    description: Control validation / reconciliation jobs
paths:
  /environment:
    get:
      tags:
        - Environment
      summary: Get all files available per environment
      operationId: getFilesByEnvironment
      responses:
        '200':
          description: Successful operation
          content:
            application/json:
              schema:
                type: array
                items:
                  $ref: '#/components/schemas/EnvironmentData'
  /job:
    get:
      tags:
        - Job
      summary: Get jobs by status
      operationId: getJobByStatus
      parameters:
        - name: status
          in: query
          required: true
          schema:
            type: string
            $ref: '#/components/schemas/JobStatus'
      responses:
        '200':
          description: Jobs by status
          content:
            application/json:
              schema:
                type: array
                schema:
                  type: object
                  $ref: '#/components/schemas/JobDetails'

    post:
      tags:
        - Job
      summary: Start a new job
      operationId: startNewJob
      requestBody:
        description: The job settings to start with
        content:
          application/json:
            schema:
              type: object
              $ref: '#/components/schemas/JobParameters'
      responses:
        '201':
          description: Job created
          content:
            application/json:
              schema:
                type: object
                $ref: '#/components/schemas/JobDetails'
        '400':
          description: Job creation failed
          content:
            application/json:
              schema:
                type: object
                $ref: '#/components/schemas/ErrorDetail'
  /job/{jobId}:
    get:
      tags:
        - Job
      summary: Get a job
      operationId: getJob
      parameters:
        - name: jobId
          in: path
          description: ID of job to action
          required: true
          schema:
            type: string
      responses:
        '200':
          description: Job details
          content:
            application/json:
              schema:
                type: object
                $ref: '#/components/schemas/JobDetails'
        '404':
          description: Job not found
          content:
            application/json:
              schema:
                type: object
                $ref: '#/components/schemas/ErrorDetail'
    put:
      tags:
        - Job
      summary: Control a job
      operationId: controlJob
      parameters:
        - name: jobId
          in: path
          description: ID of job to action
          required: true
          schema:
            type: string
        - name: action
          in: query
          required: true
          schema:
            type: string
            enum:
              - pause
              - stop
              - resume
      responses:
        '200':
          description: Job control
          content:
            application/json:
              schema:
                type: object
                $ref: '#/components/schemas/JobDetails'
        '404':
          description: Job not found
          content:
            application/json:
              schema:
                type: object
                $ref: '#/components/schemas/ErrorDetail'
components:
  schemas:
    EnvironmentAndLabel:
      type: object
      properties:
        env:
          type: string
        label:
          type: string
    EnvironmentData:
      type: object
      properties:
        env:
          type: string
          description: the name of the environment for file aggregation
        labels:
          type: array
          items:
            type: string
    ErrorDetail:
      type: object
      properties:
        code:
          type: number
        message:
          type: string
                detail:
                  type: string
    JobDetails:
      type: object
      properties:
        job_id:
          type: string
        job_status:
          type: string
          schema:
            $ref: '#/components/schemas/JobStatus'
        start_timestamp:
          type: number
        last_update:
          type: number
        baseline:
          type: object
          schema:
            $ref: '#/components/schemas/EnvironmentAndLabel'
        candidate:
          type: object
          schema:
            $ref: '#/components/schemas/EnvironmentAndLabel'
        events:
          type: array
          schema:
            $ref: '#/components/schemas/JobEvent'
        results:
          type: object
          schema:
            $ref: '#/components/schemas/EnvironmentAndLabel'
    JobEvent:
      type: object
      properties:
        timestamp:
          type: number
        status:
          type: object
          schema:
            $ref: '#/components/schemas/JobStatus'

    JobStatus:
      type: string
      enum:
        - started
        - paused
        - stopped
        - completed
    JobParameters:
      type: object
      properties:
        baseline:
          type: object
          schema:
            $ref: '#/components/schemas/EnvironmentAndLabel'
        candidate:
          type: object
          schema:
            $ref: '#/components/schemas/EnvironmentAndLabel'
        report:
          type: object
          schema:
            $ref: '#/components/schemas/ReportFormat'
    ReportFormat:
          type: object
      properties:
        format:
                  type: string
          enum:
           - csv
```

Directory Listing

Request: `GET  /api/v1/environment`

Payload: empty

Response:

```
{
  {
    "env": "QA",
    "labels": [
            20250123,
            20250124
          ]
  },
  {
    "env": "IWT",
    "labels": [
            20250118,
            20250213
          ]
  },
  {
    "env": "PROD",
    "labels": [
            20250123,
            20250124
          ]
  }
}
```

Job Control

Job Creation

Request: `POST /api/v1/job`

Payload:

```
{
  "baseline": {
    "env": "PROD",
    "label": 20250123
  },

  "candidate": {
    "env": "QA",
    "label": 20250124
  },

  "report": {
    "format": "CSV"
  }
}
```

Response:

```
{
    "job_id": <uuid>,
    "job_status": <status>,
    "start_timestamp": <timestamp>,
    "last_update": <timestamp>,
         "baseline": {
      "env": "PROD",
      "label": 2020123
    },
    "candidate": {
      "env": "QA",
      "label": 20250124
    },
    "events": [],
    "results": {}
}
```

Job Actions

Request: `PUT /api/v1/job/<job_id>?action=pause|stop|resume`

Payload: empty

Response:

```
{
    "job_id": <uuid>,
    "job_status": <status>,
    "start_timestamp": <timestamp>,
    "last_update": <timestamp>,
         "baseline": {
      "env": "PROD",
      "label": 20250123
    },
    "candidate": {
      "env": "QA",
      "label": 20250124
    },
    "events": [ // changes in job status over time
      {
        "timestamp": <timestamp>,
        "status": <job_status>
      }
    ],
    "results": {}
}
```

Job Listing

Request: `GET /api/v1/job?status=<status>&page=<page_number>`

- Status is optional, by default it would be full listing
- Page number is used for pagination

Payload: empty

Response:

```
{
  [
    {
           "job_id": <uuid>,
            "job_status": <status>,
           "output_format": <format>,
            "start_timestamp": <timestamp>,
            "completed_timestamp": <timestamp>,
            "last_update": <timestamp>,
                 "baseline": {
                      "env": "PROD",
                      "label": 20250123
            },
            "candidate": {
                      "env": "QA",
                      "label": 20250124
            },
        "events": [],
        "results": {}
    }
  ]
}
```

Job Retrieval

Request: `GET /api/v1/job/<job_id>`

Payload: empty

Response:

```
{
  "job_id": <uuid>,
  "job_status": <status>,
  "start_timestamp": <timestamp>,
  "completed_timestamp": <timestamp>,
  "last_update": <timestamp>,
  "baseline": {
    "env": "PROD",
    "label": 20250123
  },
  "candidate": {
    "env": "QA",
    "label": 20250124
  },
  "events": [
    {
      "timestamp": <timestamp>,
      "status": "STARTED"
    },
    {
      "timestamp": <timestamp>,
      "status": "COMPLETED"
    }
  ],
  "results": {
      "status": <validation status>,
      "processing_time": <duration>,
      "report": {
        "id": <report_id>,
        "file_size": <size>,
        "format": <csv | pdf>,
        "file_generated_at": <timestamp>,
        "file_url": <url-to-download-file>,
                "stats": {
                        // TBD: stats around number of differences, categories, et
                }
      }
  }
}
```

CLI

The application will be deployed in a Linux environment, so a bash script (or equivalent) entry point will need to be provided to interact with the application.

All output from the CLI shall be in JSON format to allow for integration with other bash scripts and applications, the JSON response model will be identical to the REST model.

Equivalent CLI commands are required for each operation available in the REST API.

The following is the structure of CLI commands:

```
validation-tool [COMMAND] [OPTIONS]
```

Directory Listing

Command:

```
validation-tool list-environments
```

Response:

```
{
  {
    "env": "QA",
    "labels": [
            20250123,
            20250124
    ]
  },
  {
    "env": "IWT",
    "labels": [
            20250118,
            20250123
          ]
  },
  {
    "env": "PROD",
    "labels": [
            20250123,
            20250124
          ]
  }
}
```

Job Control

Job Creation

Command:

```
validation-tool create-job --baseline-env <BASELINE_ENV_VALUE> --baseline-label <BASELINE_LABEL> --candidate-env <CANDIDATE_ENV_VALUE> --candidate-label <CANDIDATE_LABEL> --report-format <FORMAT>

# Example
validation-tool create-job --baseline-env PROD --baseline-label 20240123 --candidate-env QA --candidate-label 20240123 --report-format csv
```

Response:

```
{
  "baseline": {
    "env": "PROD",
    "label": 20250123
  },

  "candidate": {
    "env": "QA",
    "label": 20250124
  },

  "report": {
    "format": "CSV"
  }
}
```

Job Actions

Command:

```
validation-tool update-job-state --job-id <ID_VALUE> --action <ACTION_VALUE>

# Example
validation-tool update-job-state --job-id dd24aaad-e7cf-4342-b6d2-5be82603e18f --action pause
```

Response:

```
{
    "job_id": <uuid>,
    "job_status": <status>,
    "start_timestamp": <timestamp>,
    "last_update": <timestamp>,
         "baseline": {
      "env": "PROD",
      "label": 20250123
    },
    "candidate": {
      "env": "QA",
      "label": 20250124
    },
    "events": [ // changes in job status over time
      {
        "timestamp": <timestamp>,
        "status": <job_status>
      }
    ],
    "results": {}
}
```

Job Listing

Command:

```
validation-tool get-job --status <STATUS_VALUE> --page-number <PAGE_NUMBER_VALUE>

# Example
validation-tool get-job --status active
```

Response:

```
{
  [
    {
           "job_id": <uuid>,
            "job_status": <status>,
           "output_format": <format>,
            "start_timestamp": <timestamp>,
            "completed_timestamp": <timestamp>,
            "last_update": <timestamp>,
                 "baseline": {
                      "env": "PROD",
                      "label": 20250123
            },
            "candidate": {
                      "env": "QA",
                      "label": 20250124
            },
        "events": [],
        "results": {}
    }
  ]
}
```

Job Retrieval

Command:

```
validation-tool get-job --job-id <ID_VALUE>

# Example
validation-tool get-job --job-id dd24aaad-e7cf-4342-b6d2-5be82603e18f
```

Response:

```
{
  "job_id": <uuid>,
  "job_status": <status>,
  "start_timestamp": <timestamp>,
  "completed_timestamp": <timestamp>,
  "last_update": <timestamp>,
  "baseline": {
    "env": "PROD",
    "label": 20250123
  },
  "candidate": {
    "env": "QA",
    "label": 20250124
  },
  "events": [
    {
      "timestamp": <timestamp>,
      "status": "STARTED"
    },
    {
      "timestamp": <timestamp>,
      "status": "COMPLETED"
    }
  ],
  "results": {
      "status": <validation status>,
      "processing_time": <duration>,
      "report": {
        "id": <report_id>,
        "file_size": <size>,
        "format": <csv | pdf>,
        "file_generated_at": <timestamp>,
        "file_url": <url-to-download-file>,
                "stats": {
                        // TBD: stats around number of differences, categories, et
                }
      }
  }
}
```

Acceptance Criteria

The following are the acceptance criteria that will be used to gauge the outcome from this engagement.

Usability

- [ ] Implementation is in accordance with the HLD
- [ ] REST and CLI Interfaces available

Operational

- [ ]

- [ ] Order, Trade and ClearingTrade message types within DD files can be processed and compared
- [ ] Basic support for ITCH message comparison and reporting
- [ ] Detailed difference reports are generated in CSV format

Code Quality & Maintainability

- [ ] Code must follow SOLID principles
- [ ] Project must maintain a modular architecture, separation of concerns around CLI, logging, execution, reporting, etc
- [ ] Sufficient level of inline documentation highlighting purpose and usage
- [ ] At least 85% unit test coverage with integration tests for each major feature/requirement
- [ ] Code is stored on ASX infrastructure, in git with a CI/CD pipeline publishing to nexus
- [ ] Code quality reports run in CI builds, with 0 blocker / critical issues in sonarqube reports (or equivalent)

Performance & Efficiency

- [ ] The tool can be run on standard ASX linux servers for development and testing
- [ ] The tool can run comparisons and produce reports for ~100M messages / DD file / environment in under 10 minutes

Deliverables

- [ ] A working prototype of the Validation Tool
- [ ] Source code compliant with the aforementioned criteria
- [ ] Documentation covering setup, configuration and usage
- [ ] A comprehensive postman collection for all API interactions
- [ ] A demo showcasing execution and reporting