Step 4: Execute the Tests

Purpose and Importance

Now it's time to run your carefully prepared test names through the screening system and observe what happens. This step puts theory into practice and generates the evidence you'll analyse. Proper execution is important to ensure the results are reliable and reflective of the system's true performance. Think of this as a controlled experiment, you want to introduce the test data in a way that mimics normal operation as closely as possible, without causing unintended side effects. The FCA has gone so far as developing an "SST" (Sanctions Screening Tool) to test firms' systems with synthetic data, by executing your own tests, you're effectively doing what regulators might do externally. Good test execution entails using the right environment, carefully logging results and not altering any variables mid-test. The outcome will be a set of results (alerts triggered or not) for each test case, which forms the basis for analysis in the next step.

Key Activities and Decisions:

During test execution, your team should:

- **Use a safe testing environment**: Ideally, run tests in a non-production environment (like UAT or a sandbox) that mirrors production settings. This isolates the test from real business processes. Ensure this environment has the same configuration (lists, parameters) as production, if not, sync it first. If a separate environment is unavailable, you can use production with extreme caution, perhaps after hours, and ensure test entries are clearly flagged as such (to avoid confusing live monitoring or triggering real actions).
- **Load the test data into the system**: Depending on system capabilities, there are a few methods:
    - **Batch upload**: Many screening systems allow bulk input (e.g. uploading a file of names or a dummy customer file). This is efficient and reduces human error. Use your prepared dataset file for this if supported.
    - **Manual entry**: If batch isn't possible, you might manually enter cases one by one (tedious and risk of typos, double-check each entry aligns with your test list).
    - **Automated script**: If you have API access or can write a script, this can feed names in systematically and even capture output directly.

Whichever method, ensure all test cases are entered. If doing manually, tick them off your list as you go. It's easy to accidentally skip one or input a slightly different spelling, vigilance here preserves the integrity of the test.

- **Isolate test from live workflow**: If in production or using integrated systems, take steps to prevent test alerts from mixing with real alerts. For example, coordinate with the alerts handling team, tell them "alerts with reference containing 'TEST' are dummy for this exercise." Some systems let you set a "test mode" or route test alerts to a separate queue, use that if available. The goal is to avoid any confusion or wasting analysts' time on fake alerts and to ensure you can identify the test alerts easily.
- **Monitor the system during execution**: Have someone watch the system's behaviour as the test runs. Look out for any errors or unexpected behaviour (e.g. system crash, extremely slow processing or perhaps an

internal threshold that stops processing after X alerts). Monitoring also helps catch if, no alerts at all are coming (which might indicate a misconfiguration like the system not actually screening the input source).

- **Capture all results data**: This is crucial. As the system processes the names, it will produce outcomes, some will match (alerts) and some won't. You need to record which test name resulted in what:
    - If the system generates an alert or a log for a match, save that information. Export an alert report if possible, listing alert details. If no export, screenshot or manually note the alert exactly as it appears (including which watchlist entry it matched and any match score or reasons).
    - For test names that do not trigger an alert, note that as well. This might involve checking logs or the absence of an alert in the queue. Essentially, after running all names, reconcile which names triggered alerts, any name on your list that didn't produce one is a "no hit" result.
    - Ensure timestamps or identifiers link back to your input if possible (some systems might assign an alert ID or case number, record those next to the test name).
- **Avoid changes mid-test**: It's tempting if you see something odd to tweak a setting right away ("Oh, it missed X, let's quickly lower the threshold and rerun that one"). Resist doing this in the main execution phase. Consistency is key, you want all test cases run under the same configuration (the baseline configuration from Step 1) to accurately gauge performance. Changes come in a later step. The only reason to stop and adjust mid-run is if you discover a critical error (like the system wasn't actually using the right list, in which case, you may need to fix that and restart the entire test from scratch, not just continue with half the data on a new config).
- **Complete the run and backup results**: Make sure all test cases have been processed. If the system produces an output file or summary, save it. Save screenshots of any alerts or logs that are key evidence. Essentially, gather a complete record of "System outputs for each input." This will likely be a combination of exported data and manual notes. Store these securely, these will be your raw data for analysis.

Practical Tips and Examples:

- **Labelling test entries**: If possible, input a marker with test names (for instance, enter them as "Smith_TEST" or use a dummy account number). This ensures they stand out. For example, some firms prepend "ZZZ" to test customer names so they sort from the bottom and clearly are not real. Just be careful, adding extra text like "_TEST" could itself affect matching (it might interfere or cause a false negative if the system tries to match "Smith_TEST"). One trick, use a customer ID field or reference field for the marker instead of altering the name itself.
- **Run in small batches**: If you have a large dataset and uncertain system capacity, run in smaller batches (e.g. 20 names at a time). This way, if something goes wrong mid-way, you only have to redo a portion. It also avoids overloading the system, which could skew results or cause crashes.
- **Use system logging**: Turn on debug or audit logs if available. Some systems can log every matching decision (why it matched or not). If yours does, enabling that can provide rich information for analysis. Just ensure log settings don't change performance.
- **Example outcome recording**: Suppose you input 100 test names. The system raised 60 alerts. You export an alerts list which shows, for each alert, the name matched, the list entry matched and maybe a score. For

the 40 non-alerted names, you have to note them from your input list, maybe highlight those 40 in your spreadsheet as "no alert." Now you have a full mapping: 60 yielded alerts (with details) and 40 did not.

- **Time stamping**: Note the date/time of execution. It might be relevant to tie to which list versions were in effect. E.g. "Test executed on 17 Jan 2026 against system configuration as of that date (Lists updated through 15 Jan 2026)." Then if someone asks later, you can confirm if any list updates happened after that which were not included.
- **Prevent notifications**: Ensure the system does not send out any automated external communications due to test alerts. Some systems might have features like automatically emailing a sanctions officer or even regulatory reporting triggers. If these exist, disable them for testing or use a training mode. You don't want to accidentally send an alert email to a regulator about a test case!
- **Team coordination**: Have the team on standby when running the test. If something needs to be quickly addressed (like clearing the test alerts out of the queue afterwards), it's easier with everyone aware. After execution, make sure to purge or clearly mark test alerts in the system so they don't linger as "open" in dashboards or metrics, which could confuse reporting.
- **Contingency**: If the first attempt to run the test goes wrong (system issues or you realise a mistake in test data input), it's okay. Pause, fix the underlying issue, reset the environment (clear any test data) and run again. Just document that this happened (and ensure you're not analysing partial bad data).

Common Pitfalls:

- **Mixing test and real data causing confusion**: Failure to segregate test cases can result in analysts investigating fake alerts or, worse, taking action on them. Avoid this by clear marking and/or separate environment.
- **Losing track of cases**: If doing manual entry, it's easy to skip or mis-enter a name. If doing batch and some entries error out, they might not get processed. You must reconcile input vs alerts output to ensure each case got a verdict. It's a pitfall to assume "no alert means processed okay", verify that the system actually processed the name (check logs for any errors or count of records processed).
- **Inadequate result recording**: Not capturing enough detail is a big risk. For example, just noting "alert on John Smith" is good, but better to note "alert on John Smith matched to John Smith on UK list (score 90)". The latter helps in analysis to differentiate if it matched the right person or a partial. Without details, you may struggle to analyse why something happened.
- **Environmental mismatch**: Running tests in an environment that doesn't mirror production (outdated lists, different settings) can lead to misleading findings. Say UAT hadn't been updated with the latest sanctions, you might think the system missed "Person X" when in reality production would catch it. Always align environments or test in production carefully.
- **Adjusting mid-test**: Changing a threshold halfway and then continuing will taint the data. You won't be able to compare results apples-to-apples. It's better to complete one full pass, note issues and plan adjustments later. Consistency is key for analysis to be valid.

- **Forgetting to clean up**: After the test, especially if in production, ensure you remove or clearly close out test alerts and any dummy records. You don't want them counted in monthly compliance metrics or lingering as unresolved alerts in systems, which could cause confusion later or trigger unnecessary investigations.
- **Not considering performance issues**: If your batch is huge and slows the system down significantly, in extreme cases in production that could affect other processes. Monitor and be ready to pause if needed (again, better in UAT). Document if any performance limitation was observed (not usually the focus of this test, but notable if the system can't handle bulk well, might be a separate issue to address).

compliance assist

Checklist – Execute the Tests:

- [ ] Environment ready: Tests are run in a controlled environment (UAT or carefully in production) that mirrors production configuration (same sanctions lists versions, same matching settings).
- [ ] Test data input method chosen: A clear method to input or upload test names is in place (batch file, manual, script) and has been sanity checked.
- [ ] All test cases processed: Each name from the test dataset has been entered and processed by the system (confirm count of inputs equals count processed, accounting for any errors).
- [ ] No configuration changes during run: The system configuration remained constant throughout the test execution (no tuning tweaks made mid-test).
- [ ] Results captured: Outcomes for each test case are recorded. Alerts generated are saved (reports/screenshots with matched entity info) and non-alert cases are noted as such.
- [ ] Alerts/test cases identifiable: Test alerts or records are clearly labelled or isolated, so they won't be confused with real data (e.g. special reference tags or separate test user account used).
- [ ] System behaviour monitored: Any anomalies during execution (e.g. errors, performance issues) were observed and logged. None of these invalidated the test, if they did, the test was restarted after correction.
- [ ] No external fallout: The test did not inadvertently trigger external communications or actions (or such features were turned off). The relevant teams were aware to ignore/handle test alerts appropriately.
- [ ] Post-test cleanup done: Test entries/alerts were removed or marked resolved in the system post-execution to avoid lingering artifacts.
- [ ] Result data secured: The raw results (export files, screenshots, logs) are saved in a secure location for analysis and audit purposes.
- [ ] Execution details noted: Date/time of test, environment used, and any pertinent system version or list version info are documented to contextualise the results.