CSC 2.2.0 Interoperability Event 2026
Test Scenarios, Descriptions & Materials

1. Purpose and Audience

This document defines a set of technical scenarios for the CSC Interoperability Event. It describes the relevant CSC API interactions, associated data model usage, and the supporting artifacts made available by wallet implementations to enable consistent execution and evaluation of these scenarios.

2. Event Context (from CSC Interoperability Event Webinar)

Event	CSC Interoperability Event 2026 (in-person technical Interoperability Event)
Dates	17–18 March 2026
Location	Bucharest, Romania
Format	Integration & peer interoperability testing; on-site issue review and wrap-up
Primary focus	Wallets act as CSC API clients and test against QTSP / RSSP implementations

Out of scope for this Interoperability Event (explicitly not tested): legal or eIDAS compliance, certification readiness, performance benchmarks, and security audits.

3. Wallet Party Scope & Assumptions

3.1 In scope (Wallet Party responsibilities)

● Wallet implements CSC API client behavior (v2.2.0) for credential discovery, authorization and signing. ● Wallet orchestrates end-user experience: consent, user authentication triggers and confirmation. ● Wallet can switch between multiple QTSP/RSSP endpoints via configuration or relying party choice. ● Wallet captures evidence (screenshots, logs, request/response traces where allowed) for interoperability reporting.

3.2 Out of scope (Wallet Party)

● Internal implementation details of QTSP/RSSP services, HSM/SCD, CA operations, or supervisory processes.
● PKI issuance policy decisions and national trust-list governance.
● Full conformity/certification evaluation (this event is interoperability-focused).

4. Architectural Baseline (wallet-centric remote signing)

The scenarios assume the wallet operates as a Driving Application (DA) on the signer's device, acting as a CSC

API client to an RSSP (which may implement signatures/signDoc and/or signatures/signHash). The supporting data model structures are used to convey signature requests, authorization details, and signing inputs.

Key roles used throughout this document:

Actor	**Role in CSC API Scenarios**
Wallet Authorization Server	Primary CSC API client and orchestration point, acting as a Driving Application (DA) and, where applicable, a Signature Creation Application (SCA). The wallet is responsible for user interaction, presentation of signing intent and consent, preparation of CSC-compliant requests, and invocation of CSC API endpoints. Where OAuth 2.0–based authorization is used, the wallet typically acts as a public client. Specific OAuth client security requirements (e.g. PKCE usage) are out of scope of CSC specifications and may be defined by applicable profiles, policies, or external standards. OAuth 2.0 Authorization Server enabling user authorization of privileged signing operations, as defined in CSC API Section 10. May be operated by the RSSP or externalized as a standalone component, depending on deployment architecture.
RSSP / QTSP Service	Remote Signing Service Provider implementing CSC API endpoints for credential management and remote signature creation. Manages signing credentials and performs signature creation via underlying cryptographic components. Authorization may be handled internally or delegated to an external Authorization Server.
SCD / SCA	Logical Signature Creation Device (SCD) and/or Signature Creation Application (SCA) involved in the signature creation process.

	The SCD performs the cryptographic signature operation and is operated by, or on behalf of, the RSSP. The SCA, where present, is responsible for preparing signing input and/or signature formatting and may be implemented either within the wallet (DA/SCA) or within the RSSP, depending on the deployment architecture. Internal design and implementation details are intentionally out of scope of CSC specifications.

Relying Party (RP)
which requirements, but does not typically invoke CSC APIs directly. In some scenarios, the entity

Party that defines the signing intent and relies on the resulting signature for legal or business purposes. The RP specifies what is to be signed and under defining the signing intent may also be the holder of the signing credential.

5.1 Scenario Catalogue (CSC API v2.2.0 focus)

These scenarios define the minimum, mandatory CSC API interoperability baseline for wallet-centric deployments. All core scenarios are directly grounded in CSC API v2.2.0 and CSC Data Model v1.0.0 and are expected to be supported by all participating implementations.

ID	Category	CSC API touchpoints	What is tested (wallet perspective)	Priority	Wallet evidence
API-01	Credential discovery	credentials/list , c redentials/info	Wallet discovers available credentials and correctly interprets credential metadata, including supported algorithms and signature qualifiers	Mandatory	Request/response traces; selected credential metadata
API-02	Authorization (OAuth)	oauth2/authorize , oauth2/token a nd/or credentials /authorize	Wallet obtains credential authorization and	manages the Signature Activation Data (SAD) lifecycle safely and correctly	Mandatory flow	Redirect

evidence; SAD handling traces

ID	Category	CSC API touchpoints	What is tested (wallet perspective)	Priority	Wallet evidence
API-03	Hash-based signing (baseline)	signatures/signH ash	Wallet signs one document hash and validates the returned signature output	Mandatory	Signature value or signed container; verification report
API-03b	Batch signing (multi-hash)	signatures/signH ash(multi-hash)	Wallet signs multiple document hashes in a single signing operation, if supported by the service	Recommended	Multi-hash request; per-hash outputs
API-03c	Multi-operation transaction	signatures/signH ash + extended transaction	Wallet performs multiple signing calls within a single authorized session, if supported	Optional	Transaction traces; SAD reuse or extension traces
API-04	Document-based signing	signatures/signD oc	Wallet submits documentData for service-side formatting and signing, if supported	Optional	Signed document; service verification
API-05	Deterministic error handling	All endpoints	Wallet maps CSC-defined error responses and HTTP status codes to consistent and actionable client behavior, and records observed inconsistencies or ambiguities in error	Optional	Executed error cases; error-to-UX mapping table

API-06	MFA / challenge flows	credentials/getChallenge(+ authorize)	Wallet handles OTP / out-of-band / in-band challenges and retries correctly	Optional	Challenge UX evidence; API traces
API-08	Data model conformance via API usage	Data Model objects in requests	Wallet populates CSC Data Model v1.0.0 objects correctly (e.g. hash, documentInfo, adesParamet	Mandatory	Example payloads; acceptanc e by

ID	Category	CSC API touchpoints	What is tested (wallet perspective)	Priority	Wallet evidence
			ers, signatureQualifier) and requests are accepted by services		multiple services
API-09	Credential management (optional)	credentials/creat e, credentials/delete	Wallet executes credential creation or deletion authorizations where services support lifecycle management	Optional	Authorization detail samples; request/response traces

6.1 Detailed Scenario Specifications: Authentication and Authorization

Each scenario below is described with preconditions, step-by-step execution, expected results, and evidence. The steps are written from the Wallet Party point of view.

API-02 – Authorization & SAD lifecycle (OAuth and/or explicit)

Objective	Validate wallet ability to obtain credential authorization and manage Signature Activation Data (SAD) securely (binding, single-use constraints, expiration).

Primary CSC API endpoints	GET /oauth2/authorize, POST /oauth2/token, POST /csc/v2/credentials/authorize (if used)
Data model objects exercised	signatureCreationApproval (if used), documentInfo/documentRepresentations (as authorization input)

Preconditions:

- Wallet has a selected credential from API-01.
- Wallet has a redirect URI registered (for OAuth).
- Test user can complete required authentication (PIN/OTP/biometrics, depending on service policy).

Test steps:

1. Initiate authorization using OAuth (authorization code flow) or explicit authorization as required by the service policy.

2. Complete user authentication and consent in the wallet UI.

3. Obtain either an access token (OAuth) and/or a SAD returned by the service authorization mechanism.

4. Verify the wallet does not persist SAD beyond its valid window and does not reuse SAD across unrelated signings.

Expected results / pass criteria:

- Authorization succeeds and returns a usable authorization artifact (access token and/or SAD).
- Wallet enforces SAD expiry and avoids reuse; wallet fails safely if SAD is expired.
- Wallet handles HTTP 302 redirects during OAuth flows correctly.

Wallet evidence to capture:

- Redirect traces (authorization request URL, callback, token exchange) with secrets redacted.
- SAD lifecycle logs: creation time, expiry time, usage count (if applicable).
- UX evidence: consent screen and authentication prompt.

API-05 – Deterministic error handling

Objective	Validate that the wallet can interpret CSC error responses consistently (HTTP status codes and JSON error bodies) and map them to actionable UI states.
Primary CSC API endpoints	All relevant endpoints (especially credentials/info, authorize, signatures/signHash, signatures/signDoc)
Data model objects exercised	N/A (error payloads)

Preconditions:

- Wallet supports capturing error responses and correlating them with the triggering user action.

Test steps:

1. Trigger representative error conditions (e.g., invalid credentialID, missing/invalid SAD, invalid hashes, expired token).

2. Capture HTTP status code and the error JSON body.

3. Confirm the wallet maps each error to a stable internal error category and user-visible

message. **Expected results / pass criteria:**

- Errors are handled without crashes or ambiguous states.
- Wallet provides clear user guidance (retry, re-authenticate, choose another credential, contact provider).

Wallet evidence to capture:

- Error catalogue table populated during the event: endpoint, HTTP status, error, error_description, wallet mapping.
- Screenshots of user-visible error messages.

API-06 – Challenge-based MFA (in-band / out-of-band)

Objective	Validate wallet behavior when the service requires OTP or other challenges to authorize a signing operation.
Primary CSC API endpoints	POST /csc/v2/credentials/getChallenge, POST /csc/v2/credentials/authorize (if used), OAuth endpoints (if used)
Data model objects exercised	N/A (implicit through API payloads)

Preconditions:

- Service indicates challenge-based factors are required for the selected credential.
- Test user can receive OTP or complete required second factor.

Test steps:

1. Call credentials/getChallenge with authObjectID indicating the desired mechanism (e.g., OTP).

2. If the service returns an in-band challenge, display it and collect user input; if out-of-band, guide the user appropriately.

3. Complete authorization to obtain SAD or access token, then proceed to a signing operation (API-

03). **Expected results / pass criteria:**

- Wallet successfully completes MFA and proceeds to signing.
- Wallet handles both 200 (challenge returned) and 204 (challenge delivered out-of-band) cases when observed.

Wallet evidence to capture:

- Challenge request/response traces (redacted).
- UX evidence of challenge handling (screenshots) and user retry path.

API-08 – Data Model Conformance via API Usage

Objective	Validate correct usage of CSC Data Model v1.0.0 through real API interactions.
Primary CSC API endpoints	All endpoints carrying CSC Data Model payloads
Data model objects exercised	Hash, documentInfo / documentData, adesParameters, signatureQualifier

Preconditions:

- Wallet constructs requests using CSC Data Model structures.

Test steps:

1. Populate Data Model objects in signing and authorization requests.

2. Submit requests to multiple independent services where possible.

Expected results / pass criteria:

- Requests are accepted and processed successfully by services.
- No service-specific adaptations are required.

Wallet evidence to capture:

- Example payloads (redacted).
- Evidence of acceptance by multiple services.

6.2 Detailed Scenario Specifications: Credential Awareness

Each scenario below is described with preconditions, step-by-step execution, expected results, and evidence. The steps are written from the Wallet Party point of view.

API-01 – Credential discovery

Objective	Validate that the wallet can discover credentials and interpret credential metadata consistently across services.
Primary CSC API endpoints	GET /csc/v2/credentials/list, POST /csc/v2/credentials/info
Data model objects exercised	signatureQualifier (indirect), signingAlgorithm (indirect)

Preconditions:

- Wallet configured with at least one QTSP/RSSP base URL and client credentials (if required).
- Service provides at least one test credential accessible to the wallet.

Test steps:

1. Call credentials/list and display available credentialID values to the user.

2. For the selected credentialID, call credentials/info.

3. Render key metadata: key algorithm options, supported signing algorithms, signature qualifier(s), and multi-signature capabilities.

4. Store the returned metadata for later scenario reuse.

Expected results / pass criteria:

- Wallet can list and select a credential without manual service-side intervention.
- Wallet correctly interprets the metadata needed for signing and authorization (e.g., supported algorithms and whether multi-sign is possible).

Wallet evidence to capture:

- Request/response traces for credentials/list and credentials/info (with sensitive values redacted).
- Screenshot or export of the credential selection UI and the interpreted metadata.

API-09 – Credential creation / deletion (optional, if supported by service)

Objective	Exercise the v2.2.0 credential management endpoints and their authorization_details usage where QTSP/RSSP supports them.
Primary CSC API endpoints	POST /csc/v2/credentials/create, POST /csc/v2/credentials/delete, OAuth endpoints for credential-creation / credential-deletion
Data model objects exercised	credentialCreationRequest, credentialDeletionRequest, certificatePolicy, subjectData

Preconditions:

- Service supports credential creation/deletion for test tenants.
- Wallet can perform OAuth authorization with authorization_details for these operations.

Test steps:

1. Initiate OAuth authorization for credential creation using authorization_details including credentialCreationRequest.

2. Call credentials/create and verify the credential is created and discoverable via credentials/list. 3. Initiate

OAuth authorization for credential deletion using authorization_details including credentialID. 4. Call

credentials/delete and verify the credential is removed (and certificate revoked if requested by the test).

Expected results / pass criteria:

- New credential is created and appears in credentials/list.
- Deleted credential is no longer listed, and service responds deterministically to attempts to use it.

Wallet evidence to capture:

- OAuth authorization_details samples (redacted).
- Before/after evidence from credentials/list.
- Deletion response and any revocation indication (if applicable).

6.3 Detailed Scenario Specifications: Wallet Signing with SCA

Each scenario below is described with preconditions, step-by-step execution, expected results, and evidence. The steps are written from the Wallet Party point of view.

API-03 – Hash-based remote signing (single hash)

Objective	Demonstrate baseline interoperability: wallet signs a document digest via signatures/signHash and verifies the result.
Primary CSC API endpoints	POST /csc/v2/signatures/signHash
Data model objects exercised	hash, signingAlgorithm, signatureQualifier (if requested)

Preconditions:

- Wallet has obtained credential authorization (API-02).
- A sample document is available locally to the wallet, and its hash can be computed deterministically.

Test steps:

1. Compute hash digest of the document to be signed using an agreed algorithm (e.g., SHA-256).

2. Prepare the signHash request, including credentialID, hashes[], signature algorithm parameters, and SAD (if required by the service).

3. Submit the request to signatures/signHash.

4. Validate the returned signature value (or signed container) using an independent verifier

tool. **Expected results / pass criteria:**

- Service returns a successful response with a signature value.
- The signature validates for the document digest and is consistent with the declared algorithm.

Wallet evidence to capture:

- Exact hash input and algorithm OID used (recorded).
- Full signHash request/response (redacted).

● Verification report (tool + output) and resulting signed artifact.

API-03b – Batch signing (multi-hash in one operation)

Objective	Validate that the wallet can submit multiple hashes in one signatures/signHash request when the credential supports multi-signature transactions.
Primary CSC API endpoints	POST /csc/v2/signatures/signHash
Data model objects exercised	hash, documentRepresentations

Preconditions:

● credentials/info indicates multi-signature transactions are supported for the selected credential.
● Two or more sample documents are available and their hashes are computed.

Test steps:

1. Compute hashes for N documents.

2. Submit a single signatures/signHash request with hashes[] containing N entries.

3. Verify each returned signature element corresponds to the expected hash

input. **Expected results / pass criteria:**

● All signatures validate; no extra signatures beyond the authorized number are produced.

Wallet evidence to capture:

● Multi-hash request/response traces (redacted).
● Verification summary per document (pass/fail).

API-03c – Multi-operation transaction (extended authorization session)

Objective	Validate that the wallet can perform multiple signing operations within a single authorized session, where the service supports extended or reusable authorization contexts.
Primary CSC API endpoints	POST /csc/v2/signatures/signHash (with extended transaction support)
Data model objects exercised	hash, signatureCreationApproval (if applicable)

Preconditions:

● credentials/info indicates support for extended or reusable authorization sessions for the selected credential.

● The wallet has obtained a valid authorization artifact (e.g. SAD or access token). ● Two or more signing operations are planned within the authorization validity window.

Test steps:

1. Obtain authorization for signing using the applicable authorization mechanism.

2. Submit an initial signatures/signHash request using the authorization artifact.

3. Submit one or more additional signatures/signHash requests within the same authorized

session. 4. Observe service behavior when authorization limits are reached or the session expires.

Expected results / pass criteria:

- Multiple signing operations complete successfully within the authorized session.
- Authorization constraints are enforced deterministically by the service.
- The wallet fails safely if authorization expires or is exhausted.

Wallet evidence to capture:

- Transaction traces showing multiple signing calls within one authorization context (redacted).
- Evidence of authorization reuse, extension, or refresh behavior.
- Verification summary for each signed output.

6.4 Detailed Scenario Specifications: Wallet Signing with SCA

API-04 – Document-based signing

Objective	Validate wallet interoperability with document-based signing, where the wallet submits document data for service-side formatting and signing.
Primary CSC API endpoints	POST /csc/v2/signatures/signDoc
Data model objects exercised	documentData, documentInfo, adesParameters, signatureQualifier

Preconditions:

- The selected service supports document-based signing via signatures/signDoc.
- A sample document is available in a supported format.
- The wallet has obtained valid authorization for signing.

Test steps:

1. Prepare the document input using documentData and associated metadata. 2. Submit a

signatures/signDoc request with the document and desired signature parameters. 3. Receive

the signed document or signing result from the service.

4. Verify the returned signed document using an independent verification tool.

Expected results / pass criteria:

● The service successfully processes the document and returns a signed result. ● The signed document validates and conforms to the requested signature parameters.

Wallet evidence to capture:

● Redacted signDoc request/response traces.
● Signed document artifact.
● Verification report confirming signature validity.

7. Materials and Deliverables

The Testing Party will provide demonstrable materials to support execution, observation, and review of CSC API scenarios. The emphasis is on working integrations, observable flows, and verifiable outcomes, rather than packaged software or source code.

ID	Artifact	Description
M-01	Wallet demo instance	A working wallet instance (local, test, or hosted) integrated with CSC APIs, used to demonstrate end-to-end signing flows live or via recorded demo.
M-02	Environment & configuration overview	High-level description of how the wallet is configured for the Interoperability Event (CSC endpoint selection, client identity, redirect URIs, feature flags).
M-03	Per-scenario execution checklist	Step-by-step checklist aligned with Section 6 scenarios, used to confirm correct execution during live or recorded demonstrations.
M-04 M-05	Sample input documents Verification approach	Test documents used in scenarios (e.g. PDF, XML, JSON where applicable), together with their expected hashes. Description of how the wallet or external tools verify returned signatures (tools used, validation level).
M-06 M-07	Trace & observation guidance Findings &	Description of how request/response traces are captured and sensitive values redacted (no full logs required). Template for recording per-scenario results, issues observed, and

	observations template	interoperability notes.

8. Evidence Package (Wallet Demonstration Evidence)

For each executed scenario, the Testing Party will provide an evidence bundle, captured during live execution or recorded demonstration, where permitted. Evidence focuses on observable behavior and outcomes.

- Scenario ID and execution timestamp
- CSC service identifier (RSSP/QTSP name and base URL)
- Screenshots or screen recordings showing consent, authentication/MFA, and signing completion or failure
- Redacted request/response payload excerpts (where useful)
- HTTP status codes and error responses for failed cases

9. Open Interoperability Observations and Feedback

During scenario execution, certain aspects may be identified as implementation-dependent, ambiguous, or inconsistent across services. These observations will be captured as structured feedback to support continuous improvement of CSC APIs.

- Clarifications needed in CSC API specifications or data model usage
- Areas where wallet implementations require additional guidance
- Observed differences in behavior across CSC-compliant services
- Suggestions for improving API usability, error semantics, authorization flows, redirect handling, or data model constraints
- Proposed follow-up items or future considerations for CSC APIs