TrustArc

Mobile Consent SDK

Implementation Guide

Revision History

Document Version	Date	Description
1.0	03-25-2026	First version of the document.

About This Document

This document provides guidance on how to integrate and implement the Mobile Consent SDK across supported platforms, including Android, iOS, React Native, and Flutter. It outlines the SDK initialization lifecycle, configuration steps, consent UI handling, callback management, and consent data processing.

The purpose of this guide is to help development teams correctly set up the SDK, ensure proper consent behavior, and support compliance requirements through consistent consent management implementation.

Target Audience

This document is intended for mobile developers, technical implementation teams, and solution engineers responsible for integrating the Mobile Consent SDK into mobile applications. It may also be useful for technical project stakeholders who need to understand the SDK integration flow, consent lifecycle behavior, and platform-specific implementation considerations.

Disclaimer

Table of Contents

Overview

The sections outlined in this document provide the instructions required to integrate and implement the TrustArc Mobile Consent SDK across supported mobile platforms, including Android, iOS, React Native, and Flutter. This guide covers the SDK initialization lifecycle, consent configuration, consent UI handling, event callbacks, and consent data management for both Standard and IAB TCF modes.

Before using this integration guide, please make sure the following items are completed:

- You have obtained the correct consent domain and SDK configuration details from TrustArc.
- You have identified the trackers or privacy-sensitive features in your mobile application that require consent-based control.
- You have ensured that the SDK dependencies are added and the required platform setup steps are completed.
- You have defined a user identification strategy if known-user consent synchronization across devices is required.

Integration Lifecycle

Scenario Flow

This high-level flow applies to all Mobile Consent SDK implementations. Refer to the platform-specific sections of this guide for exact class names and method signatures. To start the process, follow these steps:

1. Instantiate the TrustArc SDK as early as possible once a valid UI context is available.

2. Configure the required SDK parameters, including:

    - Consent domain
    - SDK mode (Standard or IAB TCF 2.2)
    - Delegates or event listeners

3. Call `start()` once during app launch, or whenever configuration needs to be refreshed.

    During initialization, the SDK retrieves domain configuration and determines whether the consent UI must be displayed.

4. Display the Consent UI if required.

    - **Android**
      In Expressed mode, the consent UI is displayed automatically by the SDK.
    - **iOS / React Native / Flutter**
      When the SDK indicates that consent UI should be shown, call openCM() (or the platform-specific equivalent) to present the consent interface.

5. Subscribe to SDK initialization and consent-change callbacks to update application behavior.

    Platform callback references:

    - **Android**
        - `onInitFinish`, `onConsent`, or `TrustArcVendorConsentViewModel`

- **iOS**
  - `TAConsentViewControllerDelegate` (for receiving consent data)
- **React Native**
  - `NativeEventEmitter` (`onSdkInitFinish`, `onConsentChanges`)
- **Flutter**
  - `subscribe()` (`onSdkInitFinish`, `onConsentChanges`)

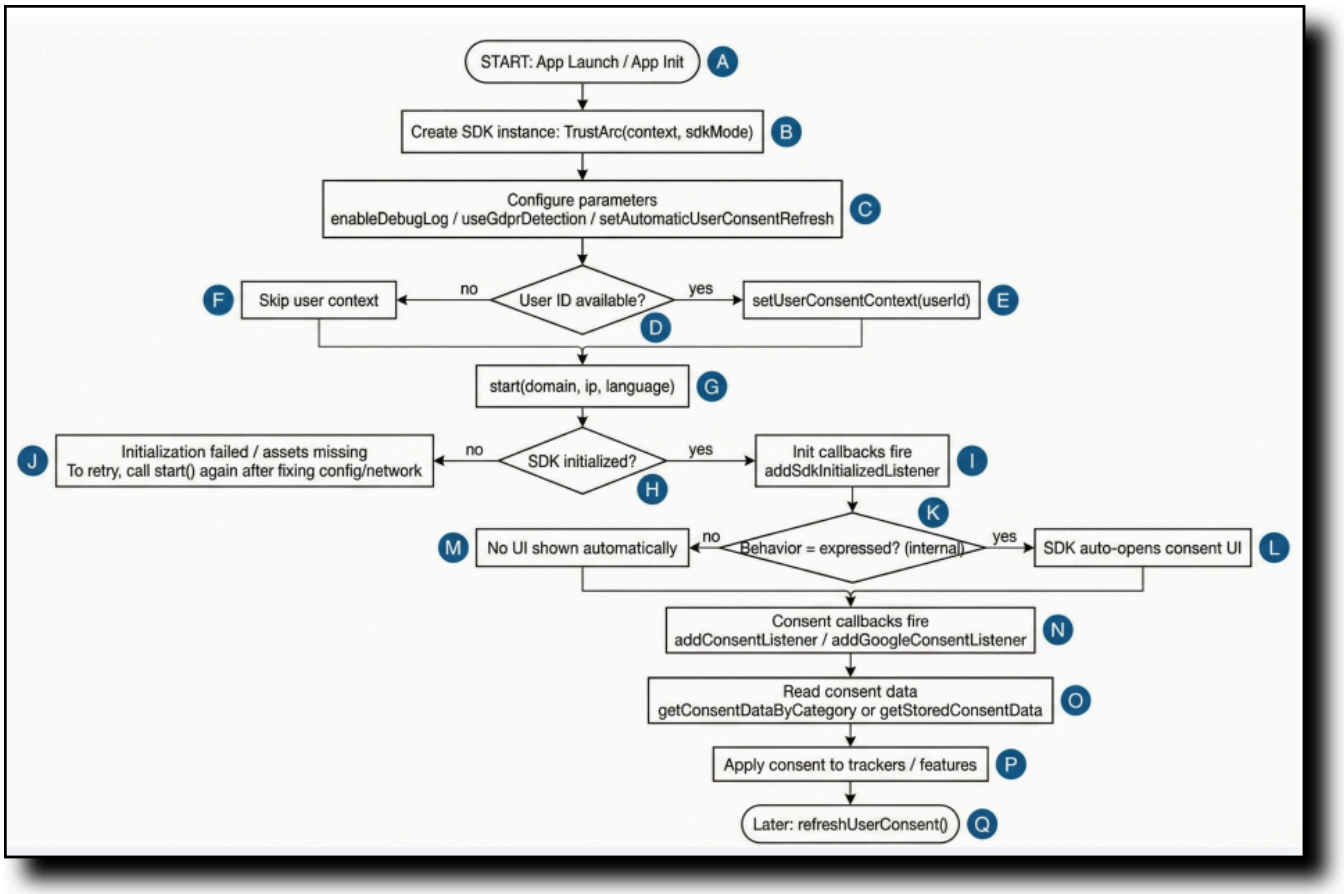
6. Retrieve consent information using **either**:

- `getStoredConsentData()` or `getConsentDataByCategory()`

Use the returned consent values to enable or disable trackers and related application features accordingly.

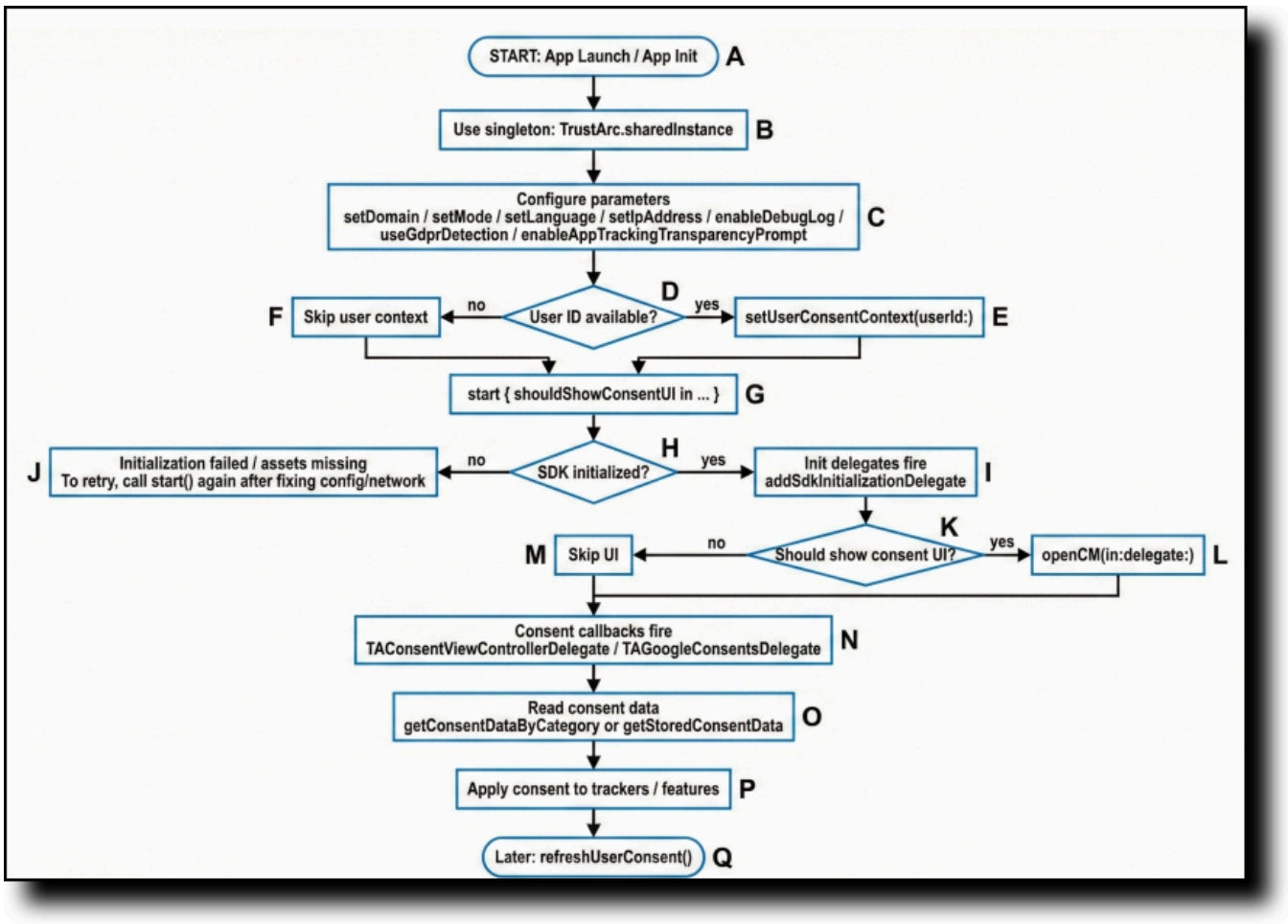
Scenario Flow Charts (By Platform)

Android



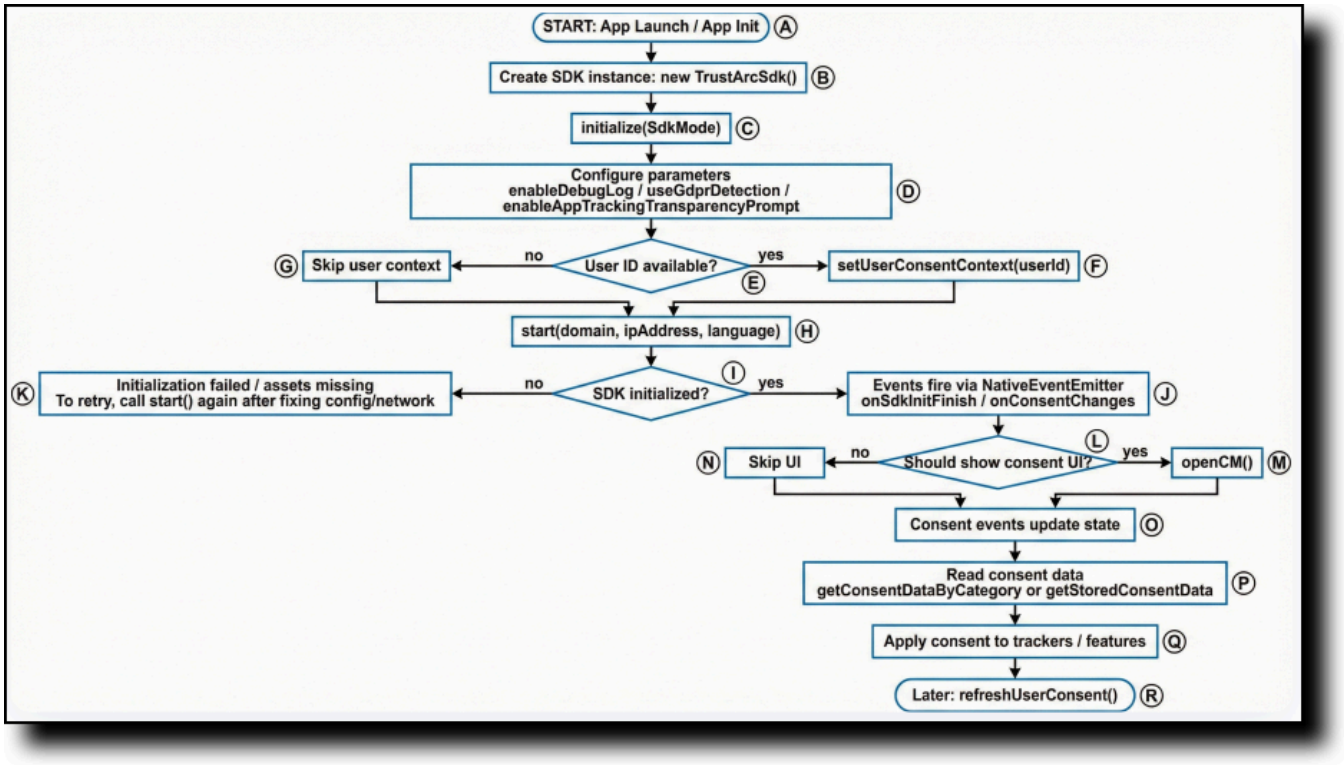
- WHEN the app launches, THEN create the SDK instance and configure parameters.
- IF a user ID is available, THEN optionally call `setUserConsentContext(userId)` to sync known-user consent; ELSE skip setting the user context.
- WHEN `start(domain, ip, language)` is called, THEN the SDK initializes.
- IF initialization succeeds, THEN initialization callbacks are triggered; ELSE initialization fails or required assets are missing — resolve configuration or network issues and retry.
- IF the consent behavior is Expressed, THEN the SDK automatically displays the consent UI; ELSE the consent UI is not shown automatically.
- WHEN consent callbacks are received, THEN read the consent data and apply it to trackers or application features. THEN optionally call `refreshUserConsent()` later if a consent refresh is required.

iOS



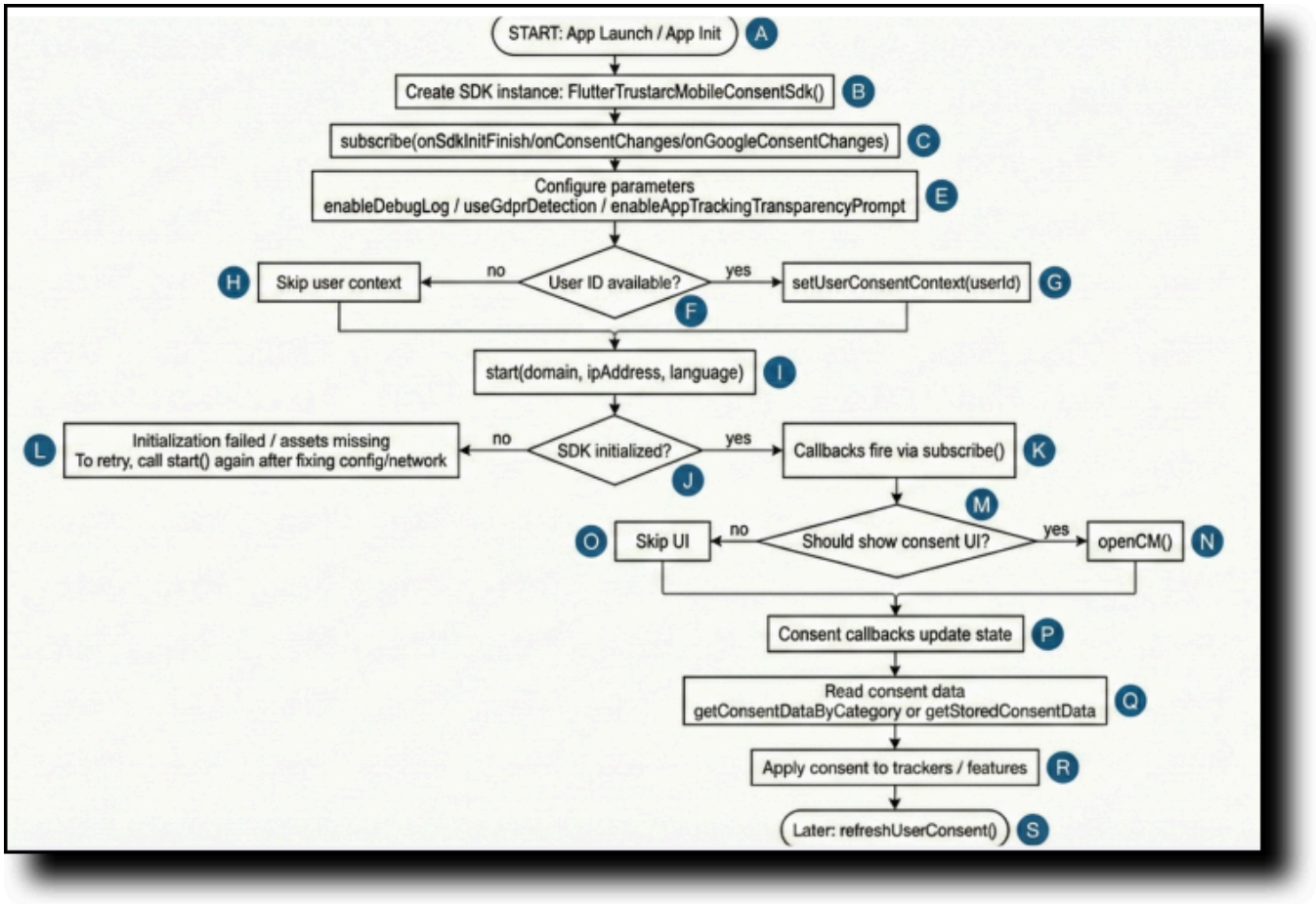
- WHEN the app launches, THEN use `TrustArc.sharedInstance` and configure the required parameters.
- IF a user ID is available, THEN optionally call `setUserConsentContext(userId:)` to synchronize known-user consent; ELSE skip setting the user context.
- WHEN `start { shouldShowConsentUI in ... }` is called, THEN the SDK initializes.
- IF initialization succeeds, THEN initialization delegates are triggered; ELSE initialization fails or required assets are missing — resolve configuration or network issues and retry.
- IF the consent UI needs to be displayed, THEN call `openCM(in:delegate:)`; ELSE skip presenting the consent UI.
- WHEN consent callbacks are received, THEN read the consent data and apply it to trackers or application features. THEN optionally call `refreshUserConsent()` later if a consent refresh is required.

React Native



- WHEN the app launches, THEN create the SDK instance and call `initialize(SdkMode)`, then configure the required parameters.
- IF a user ID is available, THEN optionally call `setUserConsentContext(userId)` to synchronize known-user consent; ELSE skip setting the user context.
- WHEN `start(domain, ipAddress, language)` is called, THEN the SDK initializes.
- IF initialization succeeds, THEN events are triggered via `NativeEventEmitter`; ELSE initialization fails or required assets are missing — resolve configuration or network issues and retry.
- IF the consent UI needs to be displayed, THEN call `openCM()`; ELSE skip presenting the consent UI.
- WHEN consent events update the application state, THEN read the consent data and apply it to trackers or application features. THEN optionally call `refreshUserConsent()` later if a consent refresh is required.

Flutter



- WHEN the app launches, THEN create the SDK instance and subscribe to the required callbacks.
- WHEN `initialize(SdkMode)` completes, THEN configure the required parameters.
- IF a user ID is available, THEN optionally call `setUserConsentContext(userId)` to synchronize known-user consent; ELSE skip setting the user context.
- WHEN `start(domain, ipAddress, language)` is called, THEN the SDK initializes.
- IF initialization succeeds, THEN subscribed callbacks are triggered; ELSE initialization fails or required assets are missing — resolve configuration or network issues and retry.
- IF the consent UI needs to be displayed, THEN call `openCM()`; ELSE skip presenting the consent

UI.

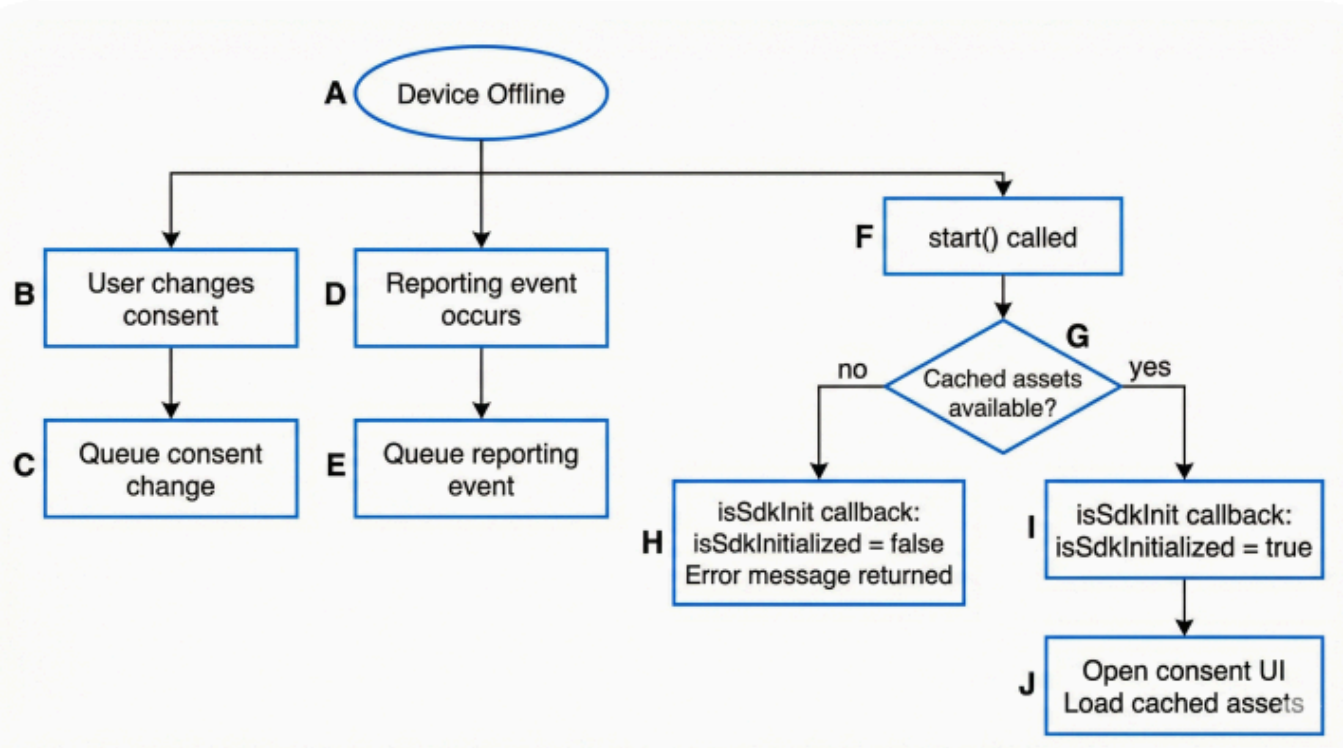
- WHEN consent callbacks update the application state, THEN read the consent data and apply it to trackers or application features. THEN optionally call `refreshUserConsent()` later if a consent refresh is required.

The following are the reasons why the order of configuration matters:

- Configuration before `start()` : the SDK uses domain/language/mode/IP to fetch the correct asset bundle and behavior; calling `start()` first can load the wrong configuration or fail initialization.
- User context before start() : `setUserConsentContext()` enables UCC sync during initialization; setting it afterward means the initial fetch is skipped and consent may be out-of-date.
- `start()` at app init: this initializes assets and determines whether UI is needed early, so you can display consent on the first screen without delay.
- UI only when needed: `start()` (or its callbacks) tells you whether to show consent UI; calling `openCM()` unconditionally can be disruptive or redundant.
- Read consent after callbacks: consent data is updated asynchronously when UI closes or background sync finishes; reading too early can return stale or empty data.

Network Down / Offline Scenario

The following information is applicable in all of the platforms:

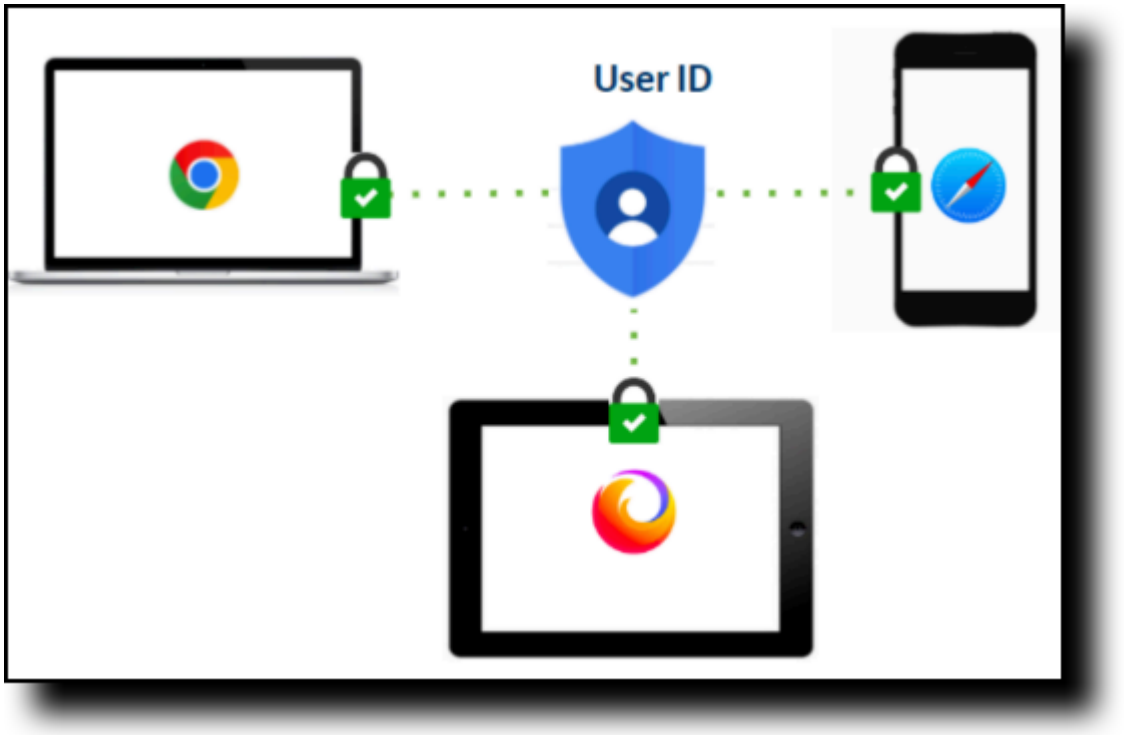


- WHEN the device is offline and the user updates consent preferences, THEN queue the consent change.
- WHEN reporting events occur while the device is offline, THEN queue the reporting events.
- WHEN `start()` is called, THEN check whether cached SDK assets are available.
- IF cached assets are available, THEN set `isSdkInitialized = true` and display the consent UI using cached assets.
- IF cached assets are not available, THEN set `isSdkInitialized = false` and return an appropriate initialization error message.

Known User Consent Implementation

Cookie Consent Manager stores and tracks consent preferences for anonymous users. However, regulatory requirements also require consent preferences to be respected across devices once a user becomes identifiable.

Known User Information



Known-user consent functionality ensures that user preferences including GPC signals and opt-out selections are consistently applied across different browsers and devices. This is achieved by associating a unique identifier with each user's stored consent preferences. Respecting the user's known and unknown choices across devices is a requirement of CPRA amendments to CCPA. Enable this functionality to ensure continued compliance.

Implementation Requirements

Before the implementation, you should do the following:

- Configure Google Tag Manager or an Autoblock solution to make this feature work. This ensures users have a smooth, non-blocking experience.
- Ensure that `gtm=1` parameter is present on your script.
- Define a `userID` identifier.
    - This can be a predefined string or a dynamic string (to remain anonymous, refer to the section below). This identifier is associated with each user who logs in to your website. Any consent preference made by the user is stored in TrustArc's database alongside the userID. This string is referenced when a user logs into their account from another device/browser to restore the last consent preference made.
    - For example: predefined `userID = abc123`

JavaScript

```
<script type="text/javascript" crossorigin

src='https://consent.trustarc.com/notice?domain=trustarc.com&userID=abc123&gtm=1'></s
cript>
```

Limitations

The CMP domain ID must match across implementations otherwise, the following are observed:

- If consent was collected under a different CMP domain, stored preferences will not be restored.
- If the CMP ID does not match the one where the user provided consent, preferences will not be restored.
- For example, if consent was recorded on Site A using CMP ID `domain=trustarc.com`, then Site B uses CMP ID `domain=truste.com`, preferences will not be restored.

Mobile Alignment

On mobile, `setUserConsentContext(userId)` must match the web `userID`. Mobile-to-web integration must be enabled using the same consent domain as the web.

Sample Flow

1. The user logs in to the website.
2. The user accepts or declines preferences from the consent banner or modal.
3. Preferences are saved to the consent center.
4. The user opens a mobile app with the same account.
5. The App fetches consents via `setUserConsentContext`.
6. The SDK fetches the preferences and loads them on the app.
7. The user does not need to re-consent; consent expiry is synced to the mobile app.

NOTE: This needs a known user flow. If there is no login flow, the consuming website or app should save the userId elsewhere. For example, generate a website userId and store it in session storage until the user logs in.

JavaScript

```
window.setTAStoreUserId = function () {

const taSessionId = `user_${Date.now()}`;

window.sessionStorage.setItem('taSessionId', taSessionId);

}

// Create user Id and store on session storage

window.setTAStoreUserId();


// Retrieve stored id and put as param on notice link

const taStoredUserId = window.sessionStorage.getItem("taSessionId");

var ccmScript = document.createElement('script');

ccmScript.setAttribute('type','text/javascript');
```

```
ccmScript.setAttribute('crossorigin','');

ccmScript.setAttribute('async','async');

ccmScript.setAttribute('src','//consent.trustarc.com/notice?domain=trustarc.com&userID='+taStoredUserId+'>m=1');

document.head.appendChild(ccmScript);
```

Common Errors and Causes

Android (trustarc-consent-sdk)

Error	Cause	Resolution
`SDK initialization failed, message=$failureMessage`	Initialization failed due to a runtime error.	Inspect the failure message and fix the underlying configuration.
`Asset download failed, domain=$domainName reason=$failureMessage`	Asset download failed for the domain (network or server error).	Check domain config and network access, then retry download.
`Asset download failed, domain=$domainName`	Asset download failed without a specific reason message.	Check logs and retry asset download.
`Asset index file missing, domain=$domainName`	The asset index is missing for the domain.	Re-download assets and verify the domain configuration.
`Asset download missing json assets, assets=$missingList`	Required JSON assets are missing after download.	Re-download assets and confirm the domain publishes all JSON files.
`Asset loading failed, error=${ex.message}`	Asset files failed to load due to IO or parse errors.	Check asset integrity and storage permissions.
`Asset download failed, asset=$fileName`	The specific asset file failed to download.	Verify asset URL and network access, then retry.
`Asset download failed, asset=$fileName error=${e.message}`	Asset download failed with an exception.	Check network error details and retry.
`Failed to copy asset from bundle: $fileName`	Copying a bundled asset to local storage failed.	Check bundle contents and storage permissions.

`Failed to copy asset from bundle, asset=$fileName`	Copying a specific asset from bundle failed.	Validate asset packaging and filesystem access.
`Failed to copy the following assets: ${failedAssets.joinToString(", ")}`	Multiple assets failed to copy during setup.	Verify all assets exist in the bundle and retry.
`Failed to parse tracker page, name=$name page=$currentPage error=${e.message}`	The tracker page JSON could not be parsed.	Validate tracker file format and version.
`Failed to parse tracker domains: ${e.message}`	The tracker domain file is malformed.	Re-download trackers and validate JSON schema.
`Failed to parse webtrackers category priorities: ${e.message}`	Category priorities JSON is invalid or missing.	Re-publish webtrackers with valid category metadata.
`Failed to parse webtrackers category name map: ${e.message}`	Category name map JSON is invalid.	Re-publish webtrackers and ensure correct schema.
`Failed to parse webtrackers category priority map: ${e.message}`	Category priority map JSON is malformed.	Regenerate the webtrackers metadata.
`Failed to clear WebView cache`	Clearing WebView cache threw an exception.	Ensure WebView is initialized and retry cache clearing.
`WebView unavailable, closing consent activity`	The device WebView provider is missing or disabled.	Install/enable a WebView provider and retry.
`Failed to read policy_control setting`	Policy configuration could not be read from storage or config.	Verify policy configuration files and storage access.
`Index file not found – assets may not be downloaded`	Consent UI assets were not downloaded or were deleted.	Re-run asset download or clear cache and reinitialize the SDK.
`Asset status: $assetStatus`	Asset download state indicates an error or incomplete assets.	Inspect asset download logs and re-fetch missing assets.

`WebView error loading resource: $url, code: $errorCode, description: $description`	The WebView failed to load a consent resource (network, TLS, or URL issue).	Verify the URL is reachable and allow network access in the app.
`WebView HTTP error loading resource: $url, status: $statusCode`	The server returned a non-2xx response for a WebView resource.	Check server availability and endpoint configuration.
`User consent update failed: ${t.message}`	The consent update network request failed.	Check network connectivity and server response details.
`User consent update failed statusCode=${response.code()}`	The consent update endpoint returned an error status.	Confirm API credentials, domain, and payload validity.
`User consent not posted; missing userId or domain`	Required identifiers are missing for consent submission.	Provide a valid userId and consent domain before posting.
`Submit Consent: Problem calling API (${t.message})`	The submit-consent API call failed.	Check network connectivity, endpoint URL, and server status.
`Submit Consent: FAILED`	The consent submission returned a failure result.	Review server response details and retry after fixing configuration.
`Cannot POST user consent: userId is blank`	User ID is required for consent submission.	Ensure a non-empty userId is set before posting consent.
`Consent Country: Problem calling API (${t.message})`	The country lookup API request failed.	Verify network connectivity and API endpoint settings.
`Consent Configs: Problem calling API (${t.message})`	The consent config API request failed.	Verify endpoint URL, credentials, and network access.
`Failed to fetch user cross consent: ${t.message}`	Cross-consent API call failed.	Check network access and server status.
`Failed to fetch user cross consent statusCode=${response.code()`	The cross-consent API returned an error status.	Validate the request and ensure correct credentials.

`}`		
`Queued user consent update failed: ${t.message}`	Network request for queued consent update failed.	Verify connectivity and retry when online.
`Queued user consent update failed statusCode=${response.code()}`	Queued consent update endpoint returned an error.	Check API response and fix request parameters.
`Queued consent report failed: ${t.message}`	Queued consent report submission failed.	Check network and server status.
`Queued consent report failed statusCode=${response.code()}`	Queued report endpoint returned an error status.	Inspect server response and correct the request.
`Failed to parse queued user consent: ${e.message}`	Queued consent payload is malformed.	Clear queued entries and re-submit consent.
`Failed to parse queued consent report: ${e.message}`	Queued report data could not be parsed.	Clear malformed queued reports and retry.
`Failed to parse web consent storage: ${e.message}`	Stored consent JSON could not be parsed.	Clear corrupted stored data and re-fetch consent state.

iOS (trustarc_consent_sdk)

Error	Cause	Resolution
`Popup date check: Invalid popup date format: [popupDateStr]`	The stored popup date does not match the expected format.	Clear the stored date or write it using the required format.
`Domain config error error=unable_to_load_config, file=domain-config.json`	The domain config file could not be loaded.	Ensure the domain_config.json asset is present and valid.
`SDK initialization failed, reason=empty_domain`	The SDK was started without a consent domain.	Pass a valid domain to start() before using the SDK.
`SDK initialization failed, message={failureMessage}`	Initialization failed due to a runtime error.	Inspect the failure message and fix the underlying configuration.
`SDK not initialized`	SDK APIs were called before initialization completed.	Call start() and wait for initialization callbacks.
`Asset download missing json assets, assets=[missingList]`	Required JSON assets are missing after download.	Re-download assets and confirm the domain publishes all JSON files.
`Unable to load asset manifest, url={self.manifestURL}, hasCachedAssets={hasCache}`	The asset manifest could not be downloaded or read.	Check the manifest URL and network connectivity.
`Asset manifest download failed statusCode={httpResponse.statusCode}`	The manifest download returned an error response.	Check the manifest endpoint and response content.
`Asset manifest download failed statusCode={httpResponse.statusCode} response={responseBody}`	The manifest endpoint returned a non-2xx status.	Verify endpoint availability and access permissions.
`Asset manifest parse error`	The downloaded manifest could not be parsed.	Validate manifest JSON format and schema.

`Failed to create trustarc assets directory, path={assetsDir.path}`	Asset directory creation failed due to permissions or invalid path.	Check filesystem permissions and available storage.
`Failed to remove asset file, path={path} error={error.localizedDescription}`	Deleting an asset file failed due to permissions or missing file.	Verify file existence and permissions before deletion.
`Failed to remove legacy asset file, path={fullPath.path} error={error.localizedDescription}`	Legacy asset cleanup failed.	Ensure the file is not locked and path is valid.
`Failed to merge tracker file, file={baseName}`	Merging tracker files failed due to IO or parse error.	Ensure tracker files are complete and valid.
`Failed to save merged tracker file, file={baseName}`	Writing the merged tracker file failed.	Check storage permissions and disk space.
`Failed to merge split files`	Split asset segments could not be combined.	Verify all segments are present and readable.
`Error transferring file, path={url.path}`	File transfer failed due to IO or permissions.	Check file access permissions and available storage.
`Error reading file, file={documentName}`	Reading a file from storage failed.	Verify file existence and access permissions.
`Error reading trackers file, file={fileName}`	The trackers file could not be read.	Ensure trackers file exists and is readable.
`Error reading data contents`	Reading binary data from disk failed.	Confirm file availability and IO permissions.
`Error copying asset, asset={assetFileName}`	Copying an asset file failed.	Check the source bundle and destination permissions.
`Error copying {asset.name} from bundle: {error}`	Copy from bundle failed due to IO or permissions.	Ensure the asset exists in the bundle and storage is writable.

`Error saving {asset.name} ({currentPage})`	Saving a paged asset file failed.	Check storage permissions and disk space.
`Could not find TAAssets.bundle path`	The TAAssets bundle is missing or not packaged.	Verify the bundle is included in the build.
`Could not create bundle, path={assetBundlePath}`	Creating the asset bundle directory failed.	Check filesystem permissions and path validity.
`User cross consent fetch failed`	The cross-consent fetch operation failed without details.	Check network connectivity and server logs.
`Network error, error=failed_to_get_interfaces`	Network interface lookup failed on the device.	Ensure the device has a valid network interface and permissions.
`Network error, error=no_interfaces_found`	No network interfaces were detected on the device.	Check network settings and connectivity.
`Unexpected IAB data, data={scriptMessage.body}`	The IAB message payload was not in the expected format.	Validate the IAB data source and update to a compatible format.
`User consent not posted; missing userId or domain`	Required identifiers are missing for consent submission.	Provide a valid userId and consent domain before posting.
`User consent update failed statusCode={statusCode} response={responseBody ?? ""}`	The consent update endpoint returned an error status with a response body.	Inspect the response body to identify server-side validation errors.
`User consent update failed response={responseBody ?? ""}`	The consent update request failed with an error response body.	Check the response body for details and correct the request.
`JSON conversion failed`	JSON serialization or deserialization failed.	Validate the JSON structure and data types.
`Unexpected IAB data, data={Self.tcOptoutDefaultDataJSON}`	The default TCF opt-out data does not match the expected schema.	Verify the TCF payload and update to a compatible version.

`Invalid URL for report endpoint, url={Environment.reportEndpointString}`	The report endpoint URL is malformed or empty.	Set a valid report endpoint URL.
`Error encoding report data`	Report payload could not be serialized.	Validate report data model and encoding options.
`Consent report network error`	The report request failed due to a network error.	Check network connectivity and endpoint availability.
`Invalid response received for consent report`	The report response was malformed or unexpected.	Inspect server response and content type.
`Consent report failed statusCode={response.statusCode} response={responseBody}`	The report endpoint returned an error with a response body.	Review the response body and fix the server-side issue.
`Consent report failed statusCode={response.statusCode}`	The report endpoint returned an error status.	Check endpoint status and request validity.

TrustArc SDK APIs (from SDK sources)

Android (TrustArc.kt)

API	Signature	Notes
start	`fun start(domainName: String, language: String? = "en") {`	Starts SDK initialization with domain (and optional IP/language). Call once per app launch after configuration.
start	`fun start(domainName: String, ipAddress: String? = null, language: String? = "en") {`	Starts SDK initialization with domain (and optional IP/language). Call once per app launch after configuration.
start	`fun start( domainName: String, language: String? = null, onConsent: Runnable?, onInitFinish: Runnable? ) {`	Deprecated: use addConsentListener/addSdkInitialized Listener; call start() without callbacks.
start	`fun start( domainName: String, ipAddress: String?, language: String? = null, onConsent: Runnable?, onInitFinish: Runnable? ) {`	Deprecated: use addConsentListener/addSdkInitialized Listener; call start() without callbacks.
openCM	`fun openCM() {`	Present the consent manager UI using the domain/config from start().
openCM	`fun openCM(domainName: String, ip: String? = null) {`	Deprecated: parameters are no longer required; domain/IP are already set via start(). Use openCM() without params.

openCMWithLanguage	`fun openCMWithLanguage(domainName : String, language: String?, ip: String? = null) {`	Deprecated: parameters are no longer required; domain/language/IP are already set via start(). Use openCM() without params.
isConsentPresent	`fun isConsentPresent(): Boolean {`	Returns true when consent preferences are stored on device.
getConsentValue	`fun getConsentValue(trackerId: String): String? {`	Returns the consent value for a tracker ID; use to enable/disable individual trackers.
getTrustArcDeviceUUID	`fun getTrustArcDeviceUUID(): String {`	Returns the SDK-generated device UUID used for consent tracking.
getStoredConsentData	`fun getStoredConsentData(): List {`	Returns the raw stored consent payload (vendor list); parse as needed.
getConsentDataByCategory	`fun getConsentDataByCategory(): Map<String, TAConsent> {`	Returns consent data grouped by category; use to gate features by category.
isCategoryConsented	`fun isCategoryConsented(categoryIndex: Int): Boolean {`	Checks consent for a category index (order matches getConsentDataByCategory).
getCategoryConsent	`fun getCategoryConsent(categoryIndex: Int): TACategoryConsent? {`	Returns category name + consent status for a category index.
getLastConsent	`fun getLastConsent(): Long {`	Returns the timestamp (ms) of the

		last consent update.
getConsentLanguage	`fun getConsentLanguage(): String? {`	Returns the language code used for consent assets.
getStoredConsentExpiry	`fun getStoredConsentExpiry(): String? {`	Returns the calculated consent expiry timestamp (ms) based on consentExpiration.
getWebScript	`fun getWebScript(): String? {`	Returns the TrustArc consent script for WebView injection (includes TCF cookie when available).
getTcfString	`fun getTcfString(): String? {`	Returns the IAB TCF string from storage.
getGoogleConsents	`fun getGoogleConsents(): String? {`	Returns Google Consent Mode values as JSON.
getBehavior	`fun getBehavior(): String? {`	Returns the detected consent behavior (expressed/implied).
useGdprDetection	`fun useGdprDetection(useGdprDetection: Boolean) {`	Enable/disable automatic GDPR detection; set before start().
enableDebugLog	`fun enableDebugLog(enabled: Boolean) {`	Enable SDK debug logging (use during development/troubleshooting).
addGoogleConsentListener	`fun addGoogleConsentListener(callback: GoogleConsentsListener) {`	Registers a listener for Google Consent Mode updates.

addConsentListener	`fun addConsentListener(callback: (Map<String, TAConsent>) -> Unit) {`	Registers a listener for consent changes (category map).
removeConsentListener	`fun removeConsentListener(callback: (Map<String, TAConsent>) -> Unit) {`	Unregisters a previously added consent listener.
addSdkInitializedListener	`fun addSdkInitializedListener(callback: () -> Unit) {`	Legacy overload; prefer (Boolean, String?) to receive init status and error message.
addSdkInitializedListener	`fun addSdkInitializedListener(callback: (Boolean) -> Unit) {`	Legacy overload; prefer (Boolean, String?) to receive init status and error message.
addSdkInitializedListener	`fun addSdkInitializedListener(callback: (Boolean, String?) -> Unit) {`	New: callback params are isInitialized (true when SDK finishes initializing) and message (error message, if any).
setUserConsent	`fun setUserConsent(data: UserConsentData, callback: Callback) {`	Posts user consent to the User Consent Center (UCC) using UserConsentData.
getUserCrossConsent	`fun getUserCrossConsent(userId: String, domain: String, callback: Callback) {`	Fetches cross-domain consent from UCC for a user/context.
setUserConsentContext	`fun setUserConsentContext(userId: String) {`	Sets UCC user context; call before start() or refreshUserConsent().

clearUserConsentContext	`fun clearUserConsentContext() {`	Clears the UCC user context (e.g., on logout).
setAutomaticUserConsentRefresh	`fun setAutomaticUserConsentRefresh(enabled: Boolean) {`	Enable/disable automatic UCC refresh during start().
setAutoAcceptAllOnImplied	`fun setAutoAcceptAllOnImplied(enabled: Boolean) {`	When behavior is implied, automatically opt in all trackers.
setForegroundActivity	`fun setForegroundActivity(activity: Activity?) {`	Sets the current Activity so background processing can surface UI if needed.
setBackgroundWebViewVisible	`fun setBackgroundWebViewVisible(enabled: Boolean) {`	Shows the hidden background WebView for debugging.
refreshUserConsent	`fun refreshUserConsent(onComplete: (() -> Unit)? = null) {`	Manually refreshes UCC consent for the current user context.
setActivity	`fun setActivity(activity: Activity) {`	Sets the current Activity reference used by the SDK (call onResume if needed).
constructor	`constructor(`	Deprecated constructor overload; use TrustArc(context, sdkMode) then addGoogleConsentListener().

Android (TASharedInstance)

API	Signature	Notes
initSdk	`fun initSdk(context: Context, sdkMode: SdkMode = Standard, googleConsentListener: GoogleConsentsListener)`	Deprecated constructor-style init; use TrustArc(context, sdkMode) then addGoogleConsentListener(...).
setSdkInstance	`fun setSdkInstance(trustArc: TrustArc)`	Sets the singleton instance.
getSdkInstance	`fun getSdkInstance(): TrustArc`	Returns the initialized SDK instance.
isSdkInitialized	`fun isSdkInitialized(): Boolean`	Returns true when the singleton is initialized and assets are ready.

iOS (TrustArc.swiftinterface)

API	Signature	Notes
didInitialized	`@objc @_Concurrency.MainActor public static let didInitialized: Swift.String`	Notification name posted when SDK initialization finishes.
controller	`@objc @_Concurrency.MainActor public var controller: trustarc_consent_sdk.TAConsentViewController`	Consent UI controller instance used to present/handle consent UI.
isInitialized	`@objc @_Concurrency.MainActor public var isInitialized: Swift.Bool {`	True when SDK initialization has completed and assets are ready.
delegate	`@objc @_Concurrency.MainActor weak public var delegate: (any trustarc_consent_sdk.TADelegate)?`	Legacy single TADelegate callback; prefer addSdkInitializationDelegate(_:).
start	`@objc @_Concurrency.MainActor public func start(withCompletion completion: @escaping @Sendable (Swift.Bool) -> Swift.Void)`	Starts the SDK; completion indicates whether to show consent UI.
start	`@objc @_Concurrency.MainActor public func start(viewController: UIKit.UIViewController, completion: @escaping @Sendable (Swift.Bool) -> Swift.Void)`	Starts SDK and allows UI presentation.
openCM	`@_Concurrency.MainActor @objc public func openCM(in viewController: UIKit.UIViewController, delegate: (any trustarc_consent_sdk.TAConsentViewControllerDelegate)?)`	Presents consent UI using config from start().
getStoredConsentData	`@objc @_Concurrency.MainActor public func getStoredConsentData() -> [Swift.String : Any]`	Returns raw stored consent payload.

isConsentPresent	`@objc @_Concurrency.MainActor public func isConsentPresent() -> Swift.Bool`	Returns true when consent exists on device.
getConsentValue	`@objc @_Concurrency.MainActor public func getConsentValue(trackerId: Swift.String) -> Swift.String`	Returns consent value for tracker.
getTrustArcDeviceUUID	`@objc @_Concurrency.MainActor public func getTrustArcDeviceUUID() -> Swift.String`	SDK device UUID.
getConsentLanguage	`@objc @_Concurrency.MainActor public func getConsentLanguage() -> Swift.String`	Language used for consent assets.
getWebScript	`@objc @_Concurrency.MainActor public func getWebScript() -> Swift.String`	TrustArc web consent script.
getTcfString	`@objc @_Concurrency.MainActor public func getTcfString() -> Swift.String`	IAB TCF string.
getGoogleConsents	`@objc @_Concurrency.MainActor public func getGoogleConsents() -> Swift.String`	Google Consent Mode JSON.
domains	`@objc public var domains: [[Swift.String : Swift.String]]?`	TAConsent per-domain values.
value	`@objc public var value: Swift.String`	TAConsent category index value.
init	`@objc override dynamic public init()`	Default initializer.
privacyPolicyLink	`@objc @_Concurrency.MainActor @preconcurrency public var privacyPolicyLink: Swift.String?`	Privacy policy URL in UI.
loaderColor	`@objc @_Concurrency.MainActor @preconcurrency public var loaderColor: UIKit.UIColor?`	Customize loader color.
sharedInstance	`@objc @_Concurrency.MainActor public static let sharedInstance:`	Singleton SDK instance.

	`trustarc_consent_sdk.TrustArc`	
locale	`@_Concurrency.MainActor public var locale: Swift.String {`	Resolved locale.
detectGeoLocation	`@objc @_Concurrency.MainActor dynamic public func detectGeoLocation(completion: @escaping @Sendable (Swift.Bool) -> Swift.Void)`	Runs GDPR geo detection.
deInitialize	`@objc @_Concurrency.MainActor dynamic public func deInitialize()`	Resets SDK initialization state.
addSdkInitializationDelegate	`@_Concurrency.MainActor @objc dynamic public func addSdkInitializationDelegate(_ delegate: any trustarc_consent_sdk.TADelegate) -> trustarc_consent_sdk.TrustArc`	Registers lifecycle delegate.
removeSdkInitializationDelegate	`@_Concurrency.MainActor @objc dynamic public func removeSdkInitializationDelegate(_ delegate: any trustarc_consent_sdk.TADelegate) -> trustarc_consent_sdk.TrustArc`	Removes lifecycle delegate.
saveConsentData	`@objc @_Concurrency.MainActor dynamic public func saveConsentData(consents: [Swift.String : Any])`	Persist consent payload manually.
saveWebConsentData	`@objc @_Concurrency.MainActor dynamic public func saveWebConsentData(consents: [Swift.String : Any])`	Persist web consent payload.
getStoredWebConsentData	`@objc @_Concurrency.MainActor dynamic public func getStoredWebConsentData() -> [Swift.String : Any]`	Returns stored web consent data.

getConsentDataByCategory	`@objc @_Concurrency.MainActor dynamic public func getConsentDataByCategory() -> [Swift.String : trustarc_consent_sdk.TAConsent]`	Consent grouped by category.
getStoredNoticePrefString	`@objc @_Concurrency.MainActor dynamic public func getStoredNoticePrefString() -> Swift.String?`	Deprecated.
getConsentValue	`@objc @_Concurrency.MainActor dynamic public func getConsentValue(tracker: Swift.String) -> Swift.String`	Returns tracker consent value.
getTCString	`@objc @_Concurrency.MainActor dynamic public func getTCString() -> Swift.String?`	Deprecated.
setMode	`@objc @_Concurrency.MainActor dynamic public func setMode(_ sdkMode: trustarc_consent_sdk.SdkMode) -> trustarc_consent_sdk.TrustArc`	Sets SDK mode.
setDomain	`@objc @_Concurrency.MainActor dynamic public func setDomain(_ domain: Swift.String) -> trustarc_consent_sdk.TrustArc`	Sets consent domain.
setIpAddress	`@objc @_Concurrency.MainActor dynamic public func setIpAddress(_ ipAddress: Swift.String) -> trustarc_consent_sdk.TrustArc`	Overrides IP for geo detection.
setDomain	`@objc @_Concurrency.MainActor dynamic public func setDomain(_ domain: Swift.String, language: Swift.String) -> trustarc_consent_sdk.TrustArc`	Sets domain + language.
setLanguage	`@objc @_Concurrency.MainActor dynamic public func setLanguage(_`	Overrides asset language.

	`language: Swift.String) -> trustarc_consent_sdk.TrustArc`	
useGdprDetection	`@objc @_Concurrency.MainActor dynamic public func useGdprDetection(_ useGdprDetection: Swift.Bool) -> trustarc_consent_sdk.TrustArc`	Deprecated.
enableAppTrackingTransparencyPrompt	`@objc @_Concurrency.MainActor dynamic public func enableAppTrackingTransparencyPrompt(_ enable: Swift.Bool) -> trustarc_consent_sdk.TrustArc`	Enable ATT prompt.
setPrivacyPolicy	`@objc @_Concurrency.MainActor dynamic public func setPrivacyPolicy(_ privacyPolicyLink: Swift.String) -> trustarc_consent_sdk.TrustArc`	Sets privacy policy link.
setIsTrackingEnabled	`@objc @_Concurrency.MainActor dynamic public func setIsTrackingEnabled(_ isTrackingEnabled: Swift.Bool) -> trustarc_consent_sdk.TrustArc`	Set tracking flag.
setReportingDelegate	`@_Concurrency.MainActor @objc dynamic public func setReportingDelegate(_ delegate: any trustarc_consent_sdk.TAConsentReporterDelegate) -> trustarc_consent_sdk.TrustArc`	Deprecated.
addReportingDelegate	`@_Concurrency.MainActor @objc dynamic public func addReportingDelegate(_ delegate: any trustarc_consent_sdk.TAConsentReporterDelegate) -> trustarc_consent_sdk.TrustArc`	Registers reporting delegate.
removeReportingDelegate	`@_Concurrency.MainActor @objc dynamic public func removeReportingDelegate(_ delegate:`	Removes reporting delegate.

	`any trustarc_consent_sdk.TAConsentReporterDelegate) -> trustarc_consent_sdk.TrustArc`	
setGoogleConsentDelegate	`@_Concurrency.MainActor @objc dynamic public func setGoogleConsentDelegate(_ delegate: any trustarc_consent_sdk.TAGoogleConsentsDelegate) -> trustarc_consent_sdk.TrustArc`	Deprecated.
addGoogleConsentDelegate	`@_Concurrency.MainActor @objc dynamic public func addGoogleConsentDelegate(_ delegate: any trustarc_consent_sdk.TAGoogleConsentsDelegate) -> trustarc_consent_sdk.TrustArc`	Registers Google consent delegate.
removeGoogleConsentDelegate	`@_Concurrency.MainActor @objc dynamic public func removeGoogleConsentDelegate(_ delegate: any trustarc_consent_sdk.TAGoogleConsentsDelegate) -> trustarc_consent_sdk.TrustArc`	Removes Google consent delegate.
setConsentViewControllerDelegate	`@_Concurrency.MainActor @objc dynamic public func setConsentViewControllerDelegate(_ delegate: any trustarc_consent_sdk.TAConsentViewControllerDelegate) -> trustarc_consent_sdk.TrustArc`	Deprecated.
addConsentViewControllerDelegate	`@_Concurrency.MainActor @objc dynamic public func addConsentViewControllerDelegate(_ delegate: any trustarc_consent_sdk.TAConsentViewControllerDelegate) -> trustarc_consent_sdk.TrustArc`	Registers consent UI delegate.

removeConsentViewControllerDelegate	`@_Concurrency.MainActor @objc dynamic public func removeConsentViewControllerDelegate(_ delegate: any trustarc_consent_sdk.TAConsentViewControllerDelegate) -> trustarc_consent_sdk.TrustArc`	Removes consent UI delegate.
setSdkInitializationDelegate	`@_Concurrency.MainActor @objc dynamic public func setSdkInitializationDelegate(_ delegate: any trustarc_consent_sdk.TADelegate) -> trustarc_consent_sdk.TrustArc`	Deprecated.
enableDebugLog	`@_Concurrency.MainActor @objc dynamic public func enableDebugLog(_ enabled: Swift.Bool) -> trustarc_consent_sdk.TrustArc`	Enable debug logging.
isDebugLogsEnabled	`@_Concurrency.MainActor @objc public static func isDebugLogsEnabled() -> Swift.Bool`	Returns debug logging state.
getIpAddress	`@objc @_Concurrency.MainActor dynamic public func getIpAddress() -> Swift.String`	Returns current IP.
getEnvironment	`@objc @_Concurrency.MainActor dynamic public func getEnvironment() -> Swift.String`	Returns SDK environment string.
getConsentUID	`@objc @_Concurrency.MainActor dynamic public func getConsentUID() -> Swift.String`	Returns consent UID.
isEu	`@objc @_Concurrency.MainActor dynamic public func isEu() -> Swift.Bool`	Returns GDPR applicability.
getPrivacyPolicyLink	`@objc @_Concurrency.MainActor dynamic public func getPrivacyPolicyLink() -> Swift.String?`	Returns privacy policy link.

confirmAllTrackers	`@objc @_Concurrency.MainActor dynamic public func confirmAllTrackers()`	Opt-in all trackers.
confirmRequiredTrackers	`@objc @_Concurrency.MainActor dynamic public func confirmRequiredTrackers(in viewController: UIKit.UIViewController, delegate: (any trustarc_consent_sdk.TAConsentViewControllerDelegate)?)`	Confirms required trackers only; may present limited UI when needed.

iOS Delegates (Protocols)

TAGoogleConsentsDelegate

Method	Signature	Notes
didUpdateConsentData	`@objc @_Concurrency.MainActor func didUpdateConsentData(consentDict: [Foundation.NSString : Swift.Bool])`	Called when Google Consent Mode values change.

TADelegate

Method	Signature	Notes
sdkIsNotInitialized	`@objc @_Concurrency.MainActor func sdkIsNotInitialized()`	Called when SDK is not initialized or reset.
sdkIsInitializing	`@objc @_Concurrency.MainActor func sdkIsInitializing()`	Called when SDK initialization begins.

sdkIsInitialized	`@objc @_Concurrency.MainActor func sdkIsInitialized()`	Called when SDK initialization completes (legacy callback).
sdkIsInitialized	`@objc @_Concurrency.MainActor optional func sdkIsInitialized(isInitialized: Bool, message: String?)`	New: `isInitialized` is true when SDK finishes initializing; `message` contains an error message, if any.
sdkIsDownloadingAssets	`@objc @_Concurrency.MainActor optional func sdkIsDownloadingAssets()`	Called when SDK begins downloading assets.
sdkDidFinishDownloadingAssets	`@objc @_Concurrency.MainActor optional func sdkDidFinishDownloadingAssets(hasFailed: Swift.Bool)`	Called when asset download finishes; `hasFailed` indicates failure.

TAConsentViewControllerDelegate

Method	Signature	Notes
consentViewController (isLoadingWebView)	`@objc @_Concurrency.MainActor func consentViewController(_ consentViewController: trustarc_consent_sdk.TAConsentViewController, isLoadingWebView webView: WebKit.WKWebView)`	Called when the consent WebView starts loading.
consentViewController (didFinishLoadingWebView)	`@objc @_Concurrency.MainActor func consentViewController(_ consentViewController: trustarc_consent_sdk.TAConsentViewController, didFinishLoadingWebView webView:`	Called when the consent WebView finishes loading.

	`WebKit.WKWebView)`	
consentViewController (didReceiveConsentData)	`@objc @_Concurrency.MainActor func consentViewController(_ consentViewController: trustarc_consent_sdk.TAConsentViewController, didReceiveConsentData consentData: [Swift.String : Any])`	Called when consent data is received (typically when UI closes).
consentViewControllerDidClose	`@objc @_Concurrency.MainActor optional func consentViewControllerDidClose(_ consentViewController: trustarc_consent_sdk.TAConsentViewController)`	Called when the consent UI is dismissed.

TAConsentReporterDelegate

Method	Signature	Notes
consentReporterWillSend	`@objc @_Concurrency.MainActor func consentReporterWillSend(report: trustarc_consent_sdk.TAConsentReportInfo)`	Called before consent report is sent.
consentReporterDidSend	`@objc @_Concurrency.MainActor func consentReporterDidSend(report: trustarc_consent_sdk.TAConsentReportInfo)`	Called after consent report is sent.
consentReporterDidFailSending	`@objc @_Concurrency.MainActor func consentReporterDidFailSending(report: trustarc_consent_sdk.TAConsentReportInfo)`	Called when consent report fails to send.

React Native

(trustarc-mobile-sdk.d.ts)

API	Signature	Notes
initialize	`initialize(sdkMode: SdkMode): Promise<void>`	Initializes the SDK and sets the mode; call before start
start	`start(domainName: string, ipAddress: string, language: string): Promise<void>`	Starts SDK initialization with domain, IP, and language; after initialize().
isSdkInitialized	`isSdkInitialized(): Promise<boolean>`	Returns true when SDK initialization has completed and assets are ready.
openCM	`openCM(): Promise<void>`	Present the consent manager UI using the domain/config from start().
isConsentPresent	`isConsentPresent(): Promise<boolean>`	Returns true when consent preferences are stored on device.
getConsentValue	`getConsentValue(trackerId: string): Promise<string>`	Returns the consent value for a tracker ID; use to enable/disable individual trackers.
getTrustArcDeviceUUID	`getTrustArcDeviceUUID(): Promise<string>`	Returns the SDK-generated device UUID used for consent tracking.
getStoredConsentData	`getStoredConsentData(): Promise<string>`	Returns stored consent data as a JSON string; parse to vendor

		consents.
getConsentLanguage	`getConsentLanguage(): Promise<string>`	Returns the language code used for consent assets.
getWebScript	`getWebScript(): Promise<string>`	Returns the TrustArc consent script for WebView injection (includes TCF cookie when available).
getTcfString	`getTcfString(): Promise<string>`	Deprecated: use `getSharedPreferences().IABTCF_TCString i`nstead.
getGoogleConsents	`getGoogleConsents(): Promise<string>`	Returns Google Consent Mode values as JSON.
getSharedPreferences	`getSharedPreferences(): Promise<{}>`	Returns native shared preferences map (includes IABTCF fields).
getIABTCFPreferences	`getIABTCFPreferences(): Promise<IABTCFPreferences>`	Parses shared preferences into a typed IABTCFPreferences object.
useGdprDetection	`useGdprDetection(enable: boolean): Promise<boolean>`	Enable/disable automatic GDPR detection; set before start().
getConsentDataByCategory	`getConsentDataByCategory(): Promise<string>`	Returns category consent data as a JSON string; parse to apply per-category gating.
getCategoryConsent	`getCategoryConsent(categoryIndex: number): Promise<string | null>`	Returns a single category consent payload; index order matches getConsentDataByCategory().

isCategoryConsented	`isCategoryConsented(categoryIndex: number): Promise<boolean>`	Checks consent for a category index (order matches getConsentDataByCategory).
getLastConsent	`getLastConsent(): Promise<number>`	Returns the timestamp (ms) of the last consent update.
getBehavior	`getBehavior(): Promise<string>`	Returns the detected consent behavior (expressed/implied).
enableDebugLog	`enableDebugLog(enabled: boolean): Promise<void>`	Enable SDK debug logging (use during development/troubleshooting).
setUserConsentContext	`setUserConsentContext(userId: string): Promise<void>`	Sets UCC user context; call before start() or refreshUserConsent().
clearUserConsentContext	`clearUserConsentContext(): Promise<void>`	Clears the UCC user context (e.g., on logout).
refreshUserConsent	`refreshUserConsent(): Promise<void>`	Manually refreshes UCC consent for the current user context.
setAutomaticUserConsentRefresh	`setAutomaticUserConsentRefresh(enabled: boolean): Promise<void>`	Enable/disable automatic UCC refresh during start().

Flutter

(flutter_trustarc_mobile_consent_sdk.dart)

API	Signature	Notes
subscribe	`void subscribe({ Function? onSdkInitFinish, Function? onConsentChanges, Function? onGoogleConsentChanges, })`	Registers event callbacks for SDK init and consent changes (call once, keep active).
getNativeData	`Future<String?> getNativeData() async`	Returns raw native consent data payload (debug/inspection).
initialize	`Future<bool> initialize(SdkMode sdkMode) async`	Initializes the SDK and sets the mode; call before start().
start	`Future<bool> start(String domainName, String ipAddress, String language) async`	Starts SDK initialization with domain, IP, and language; call after initialize().
openCM	`Future<Null> openCM() async`	Present the consent manager UI using the domain/config from start().
isConsentPresent	`Future<bool> isConsentPresent() async`	Returns true when consent preferences are stored on device.
getConsentValue	`Future<String> getConsentValue(String trackerId) async`	Returns the consent value for a tracker ID; use to enable/disable individual

		trackers.
getTrustArcDeviceUUID	`Future<String> getTrustArcDeviceUUID() async`	Returns the SDK-generated device UUID used for consent tracking.
getStoredConsentData	`Future<String> getStoredConsentData() async`	Returns stored consent data as JSON string.
getConsentLanguage	`Future<String> getConsentLanguage() async`	Returns the language code used for consent assets.
getWebScript	`Future<String> getWebScript() async`	Returns the TrustArc consent script for WebView injection.
getTcfString	`Future<String?> getTcfString() async`	Returns the IAB TCF string from storage.
getGoogleConsents	`Future<String> getGoogleConsents() async`	Returns Google Consent Mode values as JSON.
getConsentDataByCategory	`Future<String> getConsentDataByCategory() async`	Returns category consent data JSON string.
getCategoryConsent	`Future<String?> getCategoryConsent(int categoryIndex) async`	Returns category consent payload.
isCategoryConsented	`Future<bool> isCategoryConsented(int categoryIndex) async`	Checks consent for category index.
getLastConsent	`Future<int> getLastConsent() async`	Returns last consent

		timestamp.
getBehavior	`Future<String> getBehavior() async`	Returns detected consent behavior.
enableDebugLog	`Future<void> enableDebugLog(bool enabled) async`	Enable SDK debug logging.
setUserConsentContext	`Future<void> setUserConsentContext(String userId) async`	Sets UCC user context.
clearUserConsentContext	`Future<void> clearUserConsentContext() async`	Clears UCC user context.
refreshUserConsent	`Future<void> refreshUserConsent() async`	Refreshes UCC consent manually.
setAutomaticUserConsentRefresh	`Future<bool> setAutomaticUserConsentRefresh(bool enabled) async`	Enable/disable automatic UCC refresh during `start()`.
setAutoAcceptAllOnImplied	`Future<bool> setAutoAcceptAllOnImplied(bool enabled) async`	When behavior is implied, automatically opt in all trackers.
enableAppTrackingTransparencyPrompt	`Future<bool> enableAppTrackingTransparencyPrompt(bool enabled) async`	iOS only: enable/disable ATT prompt; Android is a no-op.
getTrackingAuthorizationStatus	`Future<String> getTrackingAuthorizationStatus() async`	Returns ATT status (notDetermined/restricted/denied/authorized/notApplicable).

isConsentExpired	`Future<bool> isConsentExpired() async`	Returns true if stored consent has expired.
getConsentExpiryDate	`Future<int?> getConsentExpiryDate() async`	Deprecated: Use `getStoredConsentExpiry()` instead.
getStoredConsentExpiry	`Future<int?> getStoredConsentExpiry() async`	Returns the calculated consent expiry timestamp (ms) based on consentExpiration.