Polaris SDK Documentation

> **Document DRAFT Notes**
>
> - The SDK is currently a work in progress.
> - `current_time` will likely move to `product_sku` as for payloads and core initialisation are concerned.
> - API Endpoint `/api/beta/licensing/license_keys` will likely become `/license_api/`.

1. Components

License File (`polaris.lic`)

The license file acts as the core artifact for local licensing validation.

- **Storage Location:** Stored in the standard application folders for all platforms (e.g., Application Support) under a `/Polaris/` folder.
- **Initialization:** The SDK creates a dummy `.lic` file on the first retrieval attempt if the file does not yet exist.
- **Format:** The file contents are structured in JSON.
- **Properties:** The file contains 4 top-level properties: `version`, `data`, `signed`, and `signature`.
- **Integrity & Validation:**
  - `version`: Represents the version of the JSON payload schema to ensure future backward compatibility.
  - `signed`: A Base64 representation of the `data`.
  - `signature`: The result of signing the `signed` string using a SHA256 private key. This ensures payload integrity.
  - `data`: The plain data section, allowing developers to integrate into their own legacy license systems.
- **Security:** A public key is distributed with the SDK for checking license validity. The `data` contents themselves will be encrypted using a separate key, ensuring the signature can be modified without affecting local file decryption.

Example: Partial un-encrypted license file (Created on first check)

```
{
  "version": 1,
  "data": {
    "licensed_data": "2026-04-04T15:37:19.333Z",
    "device_id": "3342341...",
    "product_sku": "your-product",
    // ... other properties
  }
}
```

Example: Full un-encrypted license file

```
{
  "version": 1,
  "data": {
    "licensed_data": "2026-04-04T15:37:19.333Z",
    "device_id": "3342341...",
    "product_sku": "your-product", // <-- product_id in future
    "license_type": 1,
    "license_key": "XXXX-XXXX-XXXX-XXXX", // <-- future removal?
    "license_id": "789123...", // <-- future addition
    "status": "active",
    "activated_date": "2026-04-04T15:37:19.333Z",
    "expiryat": "2026-04-04T15:37:19.333Z",
    "refresh_date": "2026-04-04T15:37:19.333Z",
    "last_modified_date": "2026-04-04T15:37:19.333Z",
    "active_lic": "789...abc"
    // ... other properties
  },
  "signed": "eyJkYXRlZC...",
  "signature": "alkeTbc3...="
}
```

Storage and Validation Lifecycle

Environment	Action	Description
SERVER	Serialize	The data block is minified into a JSON string.
SERVER	Encode	That string is converted into the signed Base64 representation.
SERVER	Sign	The signed Base64 representation is hashed with SHA256 and signed with the private key to create the signature hex string.
CLIENT	Encrypt	If enabled, the resulting unified layout block containing all properties is symmetrically or asymmetrically encrypted for disk storage.
CLIENT	Decrypt	File is read and decrypted (if encryption enabled).
CLIENT	Check Sign	The signature is validated against the signed Base64 representation.

2. Using the SDK

The SDK can be used as part of an all-in-one licensing solution or in headless mode combined with the API endpoints. The SDK revolves around the Core functions.

> **Note on Code Examples**
>
> Any code shown below is demonstrative PSEUDO CODE and is not strictly language-specific unless designated otherwise.

Initialization Requirements

Users require 3 key values to initialize the SDK in any mode:

- `endpoint` : The API endpoint (i.e., `https://polaris-licensing-api.com/machine`).
- `api_key` : The API key registered in Polaris Settings. **KEEP THIS SAFE.**
- `product_sku` : The product registered in Polaris (in future `product_id`).
- `developer_id` : A unique identifier for the device where the product is running. The developer decides how to retrieve this, but it must be unique per device.

Bootstrapping

```
// Instantiate core with immutable product license credentials + will-be-use mode
const core = CoreEngine(endpoint, api_key, product_sku, device_id, false)

// Instantiate core without initialization
const core = CoreEngine(endpoint, api_key, device_id, false)
```

Core Functions Available

```
/* Asked Gemma to use /api/beta/licensing/offline/ endpoint */
void AddActData(String activ, String certSrv, Uint8List certSch)

/* Check if license currently stored is valid - will create a pseudo license */
File getLicenseFile()

/* Gets the contents from a license file */
String getLicenseFileContents(bool encrypted)

/* Save a license file with the payload of your choice. */
void saveLicense(String payload, bool encrypted)

/* Check if license currently stored is valid. Will create a pseudo license.
   Calls getLicenseFile(). Runs by default when passing true to core. */
bool hasValidLicense()

/* Checks the signature for a payload as the stored license */
bool hasValidSignature(String signedPayload, String computedSignature)

/* Append a license only if partial for use'd product */
bool exportLicenseOfIdentifier(String deviceLicenseKey)

/* Import a license file */
bool importLicenseFromFile(Files sourceLicenseFile)

/* Issue heartbeat check for private (i.e., in user online */
void checkOnlineHeartbeat(Function callback)

/* Some initialized this can be used to get the current license tier.
   Designed for both things can or can be used to enable/disable product features.
   Default = 0 for lowest with other higher values decided at activation time. */
int getLicenseTierInfo()

/* Thread safe way to grab the current days onwards for license checks */
int getDaysUntilExpiry()
```

General Integration Rules

- Once Core has been instantiated in headless mode, the developer decides which core functions to use.
- There are two higher interactive components available:
  - `InteractiveScreen` Wraps `CoreEngine` and can be instantiated directly for interactive mode.
  - `InteractiveDialog` Developers can wrap this themselves in a UI of their choice. Width and Height can be specified for both.
- Both components take 2 callbacks: `onInteractionEnd` and `onInteractionStart`. The site will activate the correct bridge based on the SDK used.
- `CoreEngine` contains a function `setCertificateFailure` to modify text on the import/export offline dialog.

Custom UI Form Validation

If implementing your own license UI, submitted form data is validated server-side. Errors are returned in an `errors` array (note: error text currently verbose, will be rectified soon).

```
{
  "error": "validation_failed",
  "errors": [
    { "property": "license_key", "message": "license_key must follow format XXXX-XXXX-XXXX-XXXX" },
    { "property": "device_id", "message": "device_identifier cannot be blank" }
  ]
}
```

3. SDK Examples (JUCE)

We provide 3 main usage scenarios (a demo JUCE app will be released on GitHub):

1. Interactive Mode using `InteractiveScreen` and `getKeyboardFocusedState()` to determine trigger.
2. Headless Mode (using license file).
3. Headless Mode (no license file).

Example: JUCE License Check (Headless / Manual)

```
void runSimpleLicenseCheckTest(const juce::String& endpoint, const juce::String& slug, const juce::String& api_key,
                               const juce::String& product_sku, const juce::String& device_id)
{
    DBG ("POLARIS SDK: Initializing Manual License Check Test...");
    DBG ("==========================================================");

    // 1. Manually instantiate core
    auto core = std::make_unique<polaris_licensing_sdk::Core> (
        endpoint,
        api_key,
        product_sku,
        device_id,
        false // <-- Automatic initialization
    );

    // 2. Check if the core is valid
    DBG ("Polaris SDK Test: Checking if core is valid...");
    if (core->isValid())
    {
        DBG ("Polaris SDK Test result: A valid license is ALREADY active on this machine.");
    }
    else
    {
        DBG ("Polaris SDK Test result: No valid license found.");
    }

    DBG ("License Contents: " + core->getLicenseFileContents());
    DBG ("Days Overdue (Soft Check): " + (juce::String) core->getDaysOverdueSoft());
    DBG ("License Tier (Soft Check): " + (juce::String) core->getLicenseTierSoft());

    DBG ("==========================================================");

    juce::JUCEApplicationBase::quit();
}
```

Example: JUCE Interactive Flow

The SDK attempts to connect to a heartbeat service. Upon failure, the dialog enters offline mode allowing file export/import. In real-world use, check `getLicenseFileContents()` before showing the dialog.

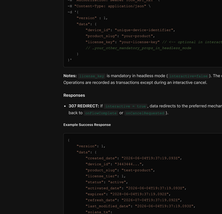
```
void runSimpleInteractiveTest(const juce::String& endpoint, const juce::String& slug, const juce::String& api_key,
                              const juce::String& product_sku, const juce::String& device_id)
{
    DBG ("==========================================================");
    DBG ("POLARIS SDK: Initializing License Test Interactive ...");
    DBG ("==========================================================");

    // 1. Instantiate core
    auto core = std::make_shared<polaris_licensing_sdk::Core> (
        endpoint,
        api_key,
        product_sku,
        device_id,
        true // <-- Automatic initialization
    );

    DBG ("Days Overdue (Soft Check): " + (juce::String) core->getDaysOverdueSoft());
    DBG ("License Tier (Soft Check): " + (juce::String) core->getLicenseTierSoft());

    licenseDialog = nullptr;
    licenseDialog = std::make_unique<polaris_licensing_sdk::DialogComponent>(core, 600, 500);

    licenseDialog->onFileComplete = [this](bool isValid) {
        DBG("Callback from License SDK: " + juce::String(isValid ? "Valid" : "Invalid"));
        if (isValid == false)
            // Treat the payload and decide what you wish to do.
            juce::JUCEApplication::quit();
    };

    licenseDialog->onClose(Dequeue = [this]() {
        DBG("Cancel signal from licensing SDK");
        if (licenseDialog != nullptr)
            // Stop receiving the licensing process
            juce::JUCEApplication::quit();
    };

    DBG ("==========================================================");
    licenseDialog->showModal();
}
```

4. API Endpoints

API Keys can be created at the account level or product level. All endpoints will deliver the following response in case of an error (Errors may or may not be returned with a 200 code).

```
{
  "error": "error_message",
  "success" : [] // <-- optional
}
```

EXAMPLE REGISTRATION DIALOG IN INTERACTIVE MODE



EXAMPLE DIALOG IF INTERACTIVE MODE TRIGGERED WITH VALID LICENSE

EXAMPLE DIALOG OFFLINE

NOTE: The text in this dialog can be changed using the setOfflineText function on the DialogComponent.

POST `/api/beta/licensing`

```
curl -X POST /api/beta/licensing/interactive/<true/false(core)>/json \
-H "Authorization: Bearer YOUR_API_KEY" \
-H "Content-Type: application/json" \
-d '{
    "version" : 1,
    "data": {
        "device_id": "unique-device-identifier",
        "product_sku": "your-product",
        "license_key": "your-license-key" // <-- optional in interactive mode
        ...your_other_mandatory_proxy_or_headless_mode
    }
}'
```

Notes: `client_id` is mandatory in headless mode (`/interactive/false`). The developer must capture and pass this value. Operations are recorded as transactions except during an interactive cancel.

Responses

- **307 REDIRECT:** If `interactive = true`, data redirects to the preferred mechanism upon completion (e.g., via JUCE webview bridge back to `getKeyboardState` or `getWebMessages()`).

Example Success Response

```
{
  "version": 1,
  "data": {
    "licensed_data": "2026-04-04T15:37:19.333Z",
    "device_id": "3342341...",
    "product_sku": "your-product",
    "license_type": 1,
    "status": "active",
    "activated_date": "2026-04-04T15:37:19.333Z",
    "expiryat": "2026-04-04T15:37:19.333Z",
    "refresh_date": "2026-04-04T15:37:19.333Z",
    "last_modified_date": "2026-04-04T15:37:19.333Z",
    "active_lic": "789...abc"
    // ... other properties
  },
  "signed": "eyJkYXRlZC...",
  "signature": "alkeTbc3...="
}
```