



Inverite Core Integration Guide

A developer guide to the Inverite integration — Bank, Bank iFrame, Age and Identity Verification, Risk, Refresh, Translator and Webhooks

The official Inverite API Documentation holds the current API parameters, request and response structures, and error codes. This guide supplements it; it is not API documentation.

<https://www.inverite.com/documentation/>

Prepared by	Operations Team
Document owner	Inverite
Version	1.0
Classification	Public
Date	2026-09-22

CONTENTS

1.	About this guide.....	2
2.	Getting started.....	2
3.	Inverite integration overview.....	3
4.	Sandbox and production.....	4
5.	Bank Verification Integration flow.....	5
6.	Product integration guides.....	6
7.	Developer implementation guidance.....	14
8.	Testing and sandbox guidance.....	16
9.	Official API documentation.....	18

1. About this guide

This guide explains how the Inverite APIs fit together and how a merchant integrates with them. It covers Bank verification, the Bank iFrame, Age and Identity Verification, Risk, Refresh, the Translator API and Webhooks. It replaces the separate implementation documents previously issued for those products.

What this guide contains

- How the Inverite products relate to each other and where each one sits in the customer journey.
- The end-to-end integration flow, including the three ways a completed request can be detected.
- Workflow diagrams, product explanations and practical implementation considerations.
- Sandbox and production concepts, and guidance on testing before go-live.

What this guide does not contain

This document is not API documentation. It does not list request parameters, response fields, schemas or error codes. Those are published in the official Inverite API Documentation and are not repeated here.

Refer to the official Inverite API Documentation for the current API parameters, request/response structures, and error codes.

<https://www.inverite.com/documentation/>

For account setup, additional sites or webhook registration, contact merchantsupport@inverite.com.

2. Getting started

2.1 Prerequisites

- An Inverite account with at least one site configured against it.
- A merchant key for the environment you are integrating against.
- For webhook delivery, an HTTPS endpoint that accepts a POST with a JSON body.

2.2 Authentication

Every API call is authenticated with your merchant key, sent as a request header. The key is different in sandbox and production, so treat it as environment configuration rather than a fixed value in your code.

Refer to the official Inverite API Documentation for the current authentication header name, format and request/response structures.

<https://www.inverite.com/documentation/>

2.3 Environments

There are two environments: sandbox, for integration and testing, and production, for live customers. Integrate and test against sandbox first. Section 4 covers the differences and what to change when you move across.

3. Inverite integration overview

The Inverite products divide into two groups, and knowing which group you are calling tells you what the integration has to handle.

Group	Products	What the integration has to handle
Customer-facing	Bank verification, Identity Verification	Presenting a session to the customer, detecting when it completes, then retrieving the result.
Server-side	Risk, Refresh, Translator	A server-to-server call against data Inverite already holds, or data you supply. The customer is not involved.

How the products relate

- Every customer-facing request is identified by a request GUID returned when the request is created. Status, Fetch, Risk and Refresh are all keyed on that GUID.
- Risk and Refresh operate on a request that has already completed successfully. Neither can run before a bank verification exists.
- Translator converts bank data held from another vendor into the Inverite format, so a request that originated elsewhere is consumed by the same downstream processing as a native verification.
- The Bank Fetch response, the Translator Fetch response and the webhook payload are built from the same response objects. One parser in your system can serve all three.

Customer journey

A typical journey runs as follows:

1. the customer reaches the point in your process where a verification is needed,
2. you create a request, the customer completes the session in the iFrame or through a link you send them,
3. your system is notified that the session finished, and
4. you retrieve the data. Any server-side scoring or refreshing happens afterwards, without the customer present.

4. Sandbox and production

Request and response structures are the same in both environments. What changes when you move to production is the base URL, your merchant key and your site IDs.

Identifiers are environment-specific

Site IDs, bank IDs and request GUIDs belong to one environment only. A sandbox site ID is not valid in production, and a GUID created in sandbox cannot be fetched from production. This is the most common cause of a working sandbox integration failing on the first production call.

Migration considerations

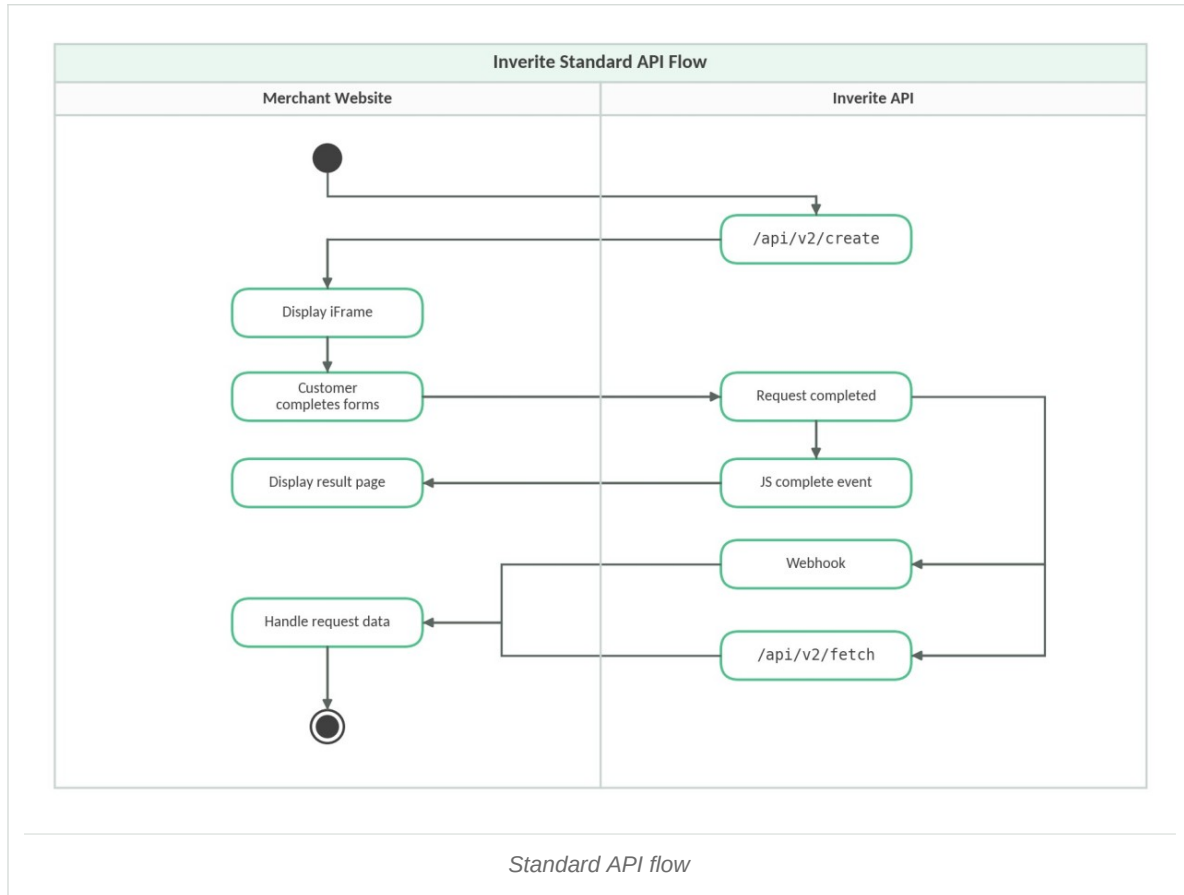
- Retrieve your site and bank IDs from the API rather than hard-coding them, so the same code works in both environments.
- Confirm your production host with your account representative.
- Register your production webhook URL before go-live; sandbox registration does not carry across.
- In sandbox, no live bank connection is made. See section 8 for how to exercise specific outcomes.

Refer to the official Inverite API Documentation for the current base URLs and endpoint reference for both environments.

<https://www.inverite.com/documentation/>

5. Bank Verification Integration flow

The flow is the same across the customer-facing products. The diagram shows the standard flow, including the three ways completion can be detected.



The four steps

- 1. Create the request.** Call Create with your site ID and your own customer reference. The response contains the request GUID and an iFrame URL. Store the GUID against your customer record.
- 2. Present the request to the customer.** Embed the returned iFrame URL in your page, or have Inverite email or text the link to the customer. Choose one: pass an email address for Inverite to notify the customer, or pass a username where the request is anonymous or you are sending the link yourself.
- 3. Detect completion.** Three mechanisms are available and only one is required. They are compared below.
- 4. Retrieve the data.** Call Fetch with the GUID. Where a webhook is configured the same data is delivered to your server and the Fetch call is not needed.

Choosing a completion mechanism

Mechanism	How it works	When it fits
JavaScript event	The iFrame posts a message to the containing page when the session completes.	Continuing the customer journey in the browser immediately, without waiting for a server-side notification. This is useful if you do not have permission to open up webhooks in your integration, or you want your UI to respond to certain events.
Webhook	Inverite posts the completed data to an HTTPS URL registered against your account.	Server-side processing. The data arrives with the notification, so no separate retrieval call is needed.
Status polling	Your server asks for the current status of one or more requests until it reports completion.	Where neither a browser event nor a registered webhook endpoint is available.

Risk and Refresh do not use this flow. They operate on a request that has already completed successfully and do not involve the customer.

Refer to the official Inverite API Documentation for the current API parameters, request/response structures, JavaScript and webhook events, and error codes.

<https://www.inverite.com/documentation/>

6. Product integration guides

Each section below covers what the product does, where it fits in the integration and any guidance specific to it. Parameters, response structures and error codes are in the online documentation.

6.1 Bank verification

Purpose

Bank verification returns a customer's accounts, transactions and income data after they authenticate with their bank.

When to use it

Wherever your process needs confirmed account details or transaction history — account ownership, income, balances — taken from the bank rather than from the customer.

High-level workflow

- Retrieve the sites configured on your account, and the supported bank list, from the API. Use these rather than hard-coded values.
- Create a request for the customer, then follow the flow in section 5.

Refer to the official Inverite API Documentation for the current request parameters, response structures, and error codes for the List, Create, Status and Fetch calls.

<https://www.inverite.com/documentation/>

6.2 Bank iFrame

Purpose

The iFrame embeds bank verification in your own web flow, typically after sign-up and information capture are complete, so the customer stays inside your journey.

Embedding

Create the request, then embed the iFrame URL returned in the response. The request GUID is returned in the same call, so no additional lookup is required before the customer starts.

Securing the iFrame URL

Supply the URL of the page the iFrame will be embedded in. Creation is then restricted to that page, which prevents unwanted requests being created against your account. The same containing-page URL is what enables the completion event, so treat it as required rather than optional.

Detecting completion

On completion the iFrame posts a message to the containing page, allowing your page to continue the customer journey without waiting for a server-side notification. Inverite publishes named lifecycle events for this purpose.

Retrieving results

Results are available in the merchant portal, or delivered to your system by webhook (section 6.7).

Refer to the official Inverite API Documentation for the current lifecycle event names, a listener example, and the Create response structure.

<https://www.inverite.com/documentation/>

6.3 Age Verification

Purpose

Age Verification confirms that a customer meets the minimum age from their government-issued ID, and that the data on the document is internally consistent. It is the right product when age is the question you need answered.

If you need to confirm that the person submitting the document is really the person it was issued to, use Identity Verification (section 6.4) instead.

Document and selfie flow

The customer submits images of the front and back of a Canadian driver's licence or provincial ID card. The barcode on the back is the authoritative data source; the front is read by OCR and cross-checked against it. A selfie can optionally be requested, and is compared by facial recognition against the photo on the front of the ID. Which of the three images are requested is set when the request is created.

Customer journey

The customer is asked for the configured images one step at a time. Each image can be dragged onto the page or selected from the device, and a link at the top of the page lets the customer send the request to their phone by SMS and photograph the ID with its camera, which is generally easier; the browser window should be left open so the session continues there once the images are submitted. Before submitting, the customer agrees to the terms and conditions and authorizes the transmission of the submitted session data to your organization; your organization is named in the consent text.

AGE

VERIFY

by



inverite

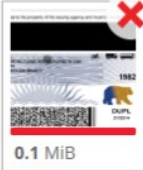
In order to verify your Photo & ID, we require the following images from you.

Click to [Complete this process from your phone](#). This will not interrupt your browser session.

1. Please Submit a Picture of your Face

2. Please Submit a picture of the front of your Driver's License or Provincial ID card

3. Please Submit a picture of the back of your Driver's License or Provincial ID card



0.1 MiB

You may Drag & Drop an image from your computer here.

Click Here To Upload/Take a Picture

By pressing "I Agree - Submit", I hereby agree to the [terms and conditions](#) and both request and authorize the transmission of all submitted session data to Test Age Verification.

I Agree - Submit

Customer capture screen

Integration flow

The flow follows section 5: create the request, embed the returned URL in your customer process, detect completion by JavaScript event or webhook, then retrieve the result. The webhook payload matches the Fetch response for this product.

Refer to the official Inverite API Documentation for the current requested-image values, Fetch response structure, and error codes.

<https://www.inverite.com/documentation/>

6.4 Identity Verification

Purpose

Identity Verification confirms that a government-issued ID is internally consistent and unexpired, and that a live person — not a photograph, a screen or a recording — presented themselves alongside it. Where Age Verification (section 6.3) answers whether the customer is old enough, this product answers whether the document and the person holding it hold up.

It is enabled for your account and requires a device with a camera.

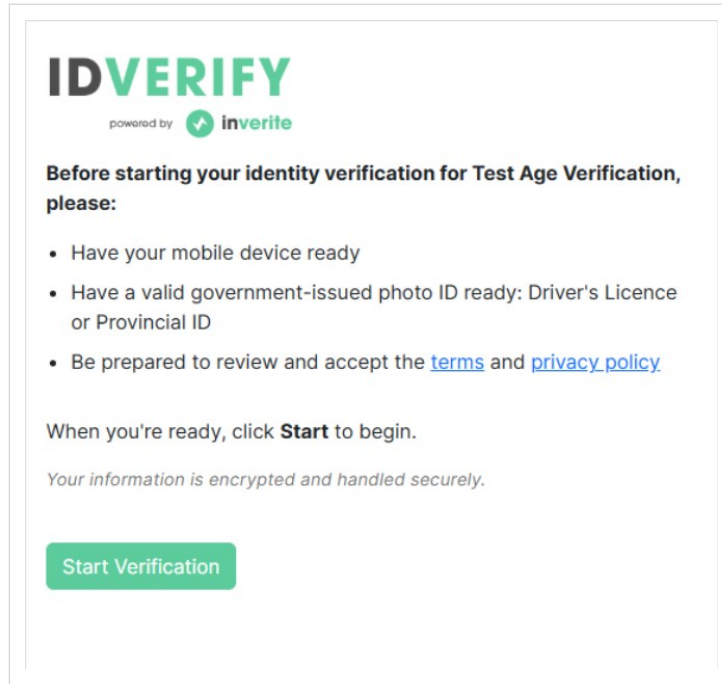
Document and face capture flow

The customer selects their document type — a driver's licence or a provincial ID card — then submits images of the front and the back, and completes a guided live face capture. All three are required; there is no configuration that omits any of them.

The barcode on the back of the document is the authoritative data source. The front is read by OCR and every field that appears on both sides is compared against it: name, date of birth, postal code, and where present the document number, issue date and expiry date. The live face capture returns a liveness confidence score.

Customer journey

The customer starts on a page listing what they will need. From a desktop browser they can send the request to their phone by SMS and complete the whole process there, which is generally easier for both photographing the ID and the face capture; the browser window should be left open so the session continues there on completion.



The screenshot shows a web interface for 'IDVERIFY', which is 'powered by inverité'. The page is titled 'Before starting your identity verification for Test Age Verification, please:'. It lists three requirements: having a mobile device ready, having a valid government-issued photo ID (Driver's Licence or Provincial ID), and being prepared to review and accept the terms and privacy policy. A green 'Start Verification' button is at the bottom. A note states 'Your information is encrypted and handled securely.'

IDVERIFY
powered by  **inverité**

Before starting your identity verification for Test Age Verification, please:

- Have your mobile device ready
- Have a valid government-issued photo ID ready: Driver's Licence or Provincial ID
- Be prepared to review and accept the [terms](#) and [privacy policy](#)

When you're ready, click **Start** to begin.

Your information is encrypted and handled securely.

Start Verification

The customer then chooses their document type and submits the front and back images, each of which can be dragged onto the page or selected from the device. The face capture runs in the browser using the device camera and guides the customer through positioning themselves in frame. Before submitting, the customer agrees to the terms and conditions and the privacy policy and authorizes the transmission of the submitted session data to your organization.



powered by 

Please submit the following to verify your identity for Test Age Verification.

Select ID Type

Driver's Licence


Face capture photo


Capture the front of your ID


Capture the back of your ID

Please submit a picture of the back of your Driver's License or Provincial ID card.

No barcode detected. Ensure high quality image in a well-lit environment and try again.

Click Here to Retake a Picture



Your privacy is our priority. All photos are kept confidential and secure, always respecting your trust.

By clicking "I Agree & Submit", I hereby agree to the [terms and conditions](#) and [privacy policy](#) and both request and authorize transmission of all submitted session data to Test Age Verification.

I Agree & Submit

Customer capture screen

Integration flow

The flow follows section 5: create the request, embed the returned URL in your customer process, detect completion by JavaScript event or webhook, then retrieve the result. The webhook payload matches the Fetch response for this product.

6.5 DefaultIQ and FlexIQ

Purpose

The Risk API scores an account that has already been verified, using data Inverite holds. The customer is not involved.

Where Risk fits

Risk runs after a successful bank verification is completed. Each successful bank verification request is allowed one risk score. A risk request fails when there is not enough account data to produce a

score, so your integration should treat a failed scoring attempt as an expected outcome rather than an error condition.

Scoring is asynchronous

Starting a scoring attempt returns a task identifier rather than a score. Poll for the result until a final state is reached; the scores are returned with the final response. Polling every five seconds is sufficient.

Requesting Risk alongside a verification

Risk can also be requested at the same time as a verification, through the add-on model, rather than as a separate call afterwards. The risk score iFrame parameter used previously is deprecated; new integrations should use the add-on model.

Risk 1.0 (Core) and Risk 2.0 (Flex)

Two risk score types are available Core and Flex. They are separate score types rather than successive versions of the same score: Flex does not replace Core, and a request can ask for either or both where your account is entitled to them.

Both can be requested as add-ons alongside a bank verification, and on a Translator submission (section 6.6). Which score types are enabled for your account, and the values each one returns, are in the online documentation.

Refer to the official Inverite API Documentation for the current Risk and Risk Status request/response structures, state values, the Core and Flex score types available as add-ons, and failure reason codes.
<https://www.inverite.com/documentation/>

6.6 Refresh

Purpose

Refresh retrieves updated data from an account the customer has already connected, using stored credentials. The customer is not involved.

Where it fits in the lifecycle

A previously successful request is required. Sending its GUID creates a new request, which progresses and completes like any other: poll for status, and a webhook is sent on completion as normal. The new request can itself be refreshed if it succeeds, so a chain of periodic refreshes is possible. Each successful request allows one refresh, so the chain advances one link at a time.

Why a refresh may fail

- The customer has changed their banking credentials.
- A previously unencountered security question was presented.
- The "save this computer" cookie has expired.

None of these are recoverable server-side. Where a refresh fails for one of these reasons, the customer has to complete a new verification.

Version 2 of the Refresh endpoint has been withdrawn. Integrations must use v3.

Refer to the official Inverite API Documentation for the current Refresh request/response structure, the available refresh options, and the applicable rate limit.

<https://www.inverite.com/documentation/>

6.6 Translator

Purpose

The Translator API converts bank data held from another vendor into the Inverite format, so that the same downstream processing can be used regardless of the data source. The currently supported vendor is Flinks.

High-level workflow

1. Submit the source data. The request is a multipart upload with two parts: a parameter file describing the request, and a data file containing the source data. A request GUID is returned.
2. Track progress with the same status mechanism used by the other products.
3. Retrieve the translated result with the same Fetch call used for a native verification. The result is built from the same response objects, with additions specific to Translator, including validation results.
4. Optional validations and scores — such as name matching, the Core and Flex risk scores, and gross income — can be requested on the same submission rather than as separate calls.

Because the translated output shares the Bank response objects, an existing bank verification integration usually needs no downstream changes to consume translated data.

Test data is provided for sandbox integration.

Refer to the official Inverite API Documentation for the current parameter file schema, upload constraints, response structures, and error codes.

<https://www.inverite.com/documentation/>

6.7 Webhooks

Purpose

On completion of a request, Inverite posts the data to a URL you provide. A webhook removes the need to poll for status and delivers the result in the same call, so for server-side integrations it is usually the simplest completion mechanism.

Setting one up

- Supply an HTTPS URL that accepts a POST with a JSON body.
- Email merchantsupport@inverite.com to have the URL registered against your account. Registration is per environment.
- Confirm your endpoint can parse a completion payload before the URL is registered (section 8).

How webhooks fit into result handling

The payload delivered to your endpoint carries the same data as the Fetch response for that product, so where a webhook is configured the separate retrieval call is not needed. Events are named per product and per lifecycle stage, which lets one endpoint handle several products.

Webhooks were previously referred to as postbacks. The two terms describe the same mechanism.

Refer to the official Inverite API Documentation for the current event names and payload structures for each product.

<https://www.inverite.com/documentation/>

7. Developer implementation guidance

7.1 Identifiers and references

- Store the request GUID returned when a request is created against your customer record. Status, Fetch, Risk and Refresh are all keyed on it.
- Pass your own customer identifier as the customer reference so that multiple requests for the same customer remain linked in your system and in the portal.
- Retrieve site and bank IDs from the API rather than hard-coding them. They differ between environments and can change as your account is configured.

7.2 Integration decisions to make up front

Decision	Options	What it affects
How the customer reaches the session	Embed the iFrame in your page, or have Inverite email or text the link.	Whether you supply a containing-page URL, and whether the browser completion event is available to you.
How you learn the session finished	JavaScript event, webhook, or status polling.	Whether you need a registered HTTPS endpoint, and whether a separate retrieval call is required.
When Risk is requested	Alongside the verification as an add-on, or as a separate call afterwards.	Whether scoring is asynchronous in your flow, and how a failed scoring attempt is handled.

7.3 Sequencing example

The example below shows the order of operations for a bank verification in PHP: create the request, store the GUID, wait for the completion notification, then fetch and store what you need. The endpoint paths and parameters are illustrative — take the current values from the online documentation.

```
<?php
$inveriteApiKey = 'my-secret-key';
$inveriteSiteId = '101';
$siteName      = 'Brand';
$isDev         = 1;

// 1. Create the request and store the GUID against your customer record
$response = inveriteCreate(1, '127.0.0.1', 'user@example.com', '6045551313');
$guid     = $response['request_guid'];

// 2. Wait for the webhook confirming the verification is complete

// 3. Fetch the data and store what your process needs
$response = inveriteFetch($guid);
$transit  = $response['accounts'][0]['transit'];
```

```
function inveriteHost()
{
    global $isDev;
    // production host: confirm with your account representative
    return $isDev == 1 ? 'sandbox.inverite.com' : 'PRODUCTION-HOST';
}

function inveriteCreate($customerId, $ip, $email = '', $phone = '')
{
    global $inveriteApiKey, $inveriteSiteId, $siteName;

    $parameters = array(
        'ip'            => $ip,
        'siteID'        => $inveriteSiteId,
        'referenceid'    => $customerId,
        'email'         => $email,
        'phone'         => $phone,
    );

    // for a phone-only notification, a unique username is required
    if ($email == '' && $phone != '')
    {
        $parameters['username'] = $siteName . '_' . $customerId;
    }

    return inveritePost("/api/v2/create", $parameters);
}

function inveriteFetch($guid)
{
    return inveriteGet("/api/v2/fetch/" . $guid);
}
?>
```

The request and fetch helpers issue a standard authenticated HTTPS call; see the online documentation for the current headers, endpoint paths and response structures.

Refer to the official Inverite API Documentation for the current endpoint paths and parameters used in this example.

<https://www.inverite.com/documentation/>

8. Testing and sandbox guidance

In sandbox, no live bank connection is made. Test accounts let you exercise a specific outcome so that your integration is tested against each path before go-live, not only the successful one.

Test approach

- Run the full flow end to end in sandbox: create, customer session, completion detection and retrieval.
- Exercise a failed verification and a failed challenge, not only the success path.
- Confirm your completion mechanism works before go-live, including webhook endpoint parsing.

- Re-test identifier handling against production values, since site IDs and GUIDs do not carry across environments.

Test accounts

Use the usernames below at the bank login screen to produce a specific outcome. Any password is accepted.

Username	Outcome
success	Successful verification, no challenge
success-multiple-accounts	Successful verification returning multiple accounts
success-question	Successful verification with a challenge question
success-multiple	Successful verification with multiple challenge questions
success-image	Successful verification with a challenge image
success-mfa	Successful verification with multi-factor authentication
fail	Failed verification

On challenge question screens, any answer is accepted except "wrong", which produces a failed challenge.

Testing a webhook endpoint

Post a sample completion payload to your own URL and confirm it can be parsed before the URL is registered against your account. Take the payload structure for the product you are integrating from the online documentation.

Translator test data

Test data is provided for sandbox integration with the Translator API.

Refer to the official Inverite API Documentation for the current test and sandbox reference detail.

<https://www.inverite.com/documentation/>

9. Official API documentation

The official Inverite API Documentation holds the current reference detail for the Inverite APIs: request parameters, request and response structures, error codes and event names. Use it wherever this guide points to it.

<https://www.inverite.com/documentation/>

For account setup, additional sites or webhook registration, contact merchantsupport@inverite.com.