NxQuickBooks Online FireDAC Driver

Getting Started Guide

Version 1.0.2 · NexCore Labs LLC · support@nxcorelabs.com · https://www.nxcorelabs.com

Table of Contents

Before You Begin — One Thing That Surprises First-Time Users

You do not need to look up a RealmID before connecting your first customer. Leave **`RealmID`** blank in your **`TFDConnection`** settings — when your customer signs in, Intuit's own consent screen handles company selection, and the driver fills in the correct **RealmID** automatically.

You only need to set it explicitly for **sandbox/development** testing, or if you're targeting a **specific company programmatically** for a user with access to more than one.

1. Introduction

The NexCore QuickBooks Online FireDAC Driver (NxFireDACQboDriverLink) is a proprietary custom FireDAC physical driver for Embarcadero RAD Studio that gives Delphi and C++Builder developers direct, native access to Intuit QuickBooks Online data using the same familiar FireDAC components they already know.

Familiar FireDAC Components

TFDConnection | TFDQuery | TFDTable | TDBGrid | TDBNavigator | FieldByName | ParamByName | First | Next | Prior | Locate | Filter | Filtered | Eof | ExecSQL | And more...

How It Works

Unlike REST wrapper libraries that require learning a new API, the NxQBO Driver works exactly like any other FireDAC database driver. Drop a TNxPhysQBODriverLink on your form, configure a TFDConnection, and your QBO company data is immediately available as a standard FireDAC dataset. Reading customers, invoices, items, and 27 other QBO entities is as simple as writing a SELECT statement, and creating or updating records works through ordinary parameterized INSERT and UPDATE statements — the same patterns you already use with SQL Server, PostgreSQL, or any other FireDAC-supported database.

The driver goes a step further for live-bound editing. Behind the scenes, a built-in command generator intercepts FireDAC's standard Edit/Append/Delete/Post mechanism and automatically translates each change into the correct QuickBooks Online write — no manual SQL required. After a successful Post, the driver also refreshes the record's SyncToken directly on the row, so a second edit can be made immediately without re-querying. This works for standard flat fields; QBO entities with nested objects or array data (such as an Invoice's line items) still require an explicit parameterized SQL statement, covered in Section 7.

OAuth2 authentication with Intuit is handled automatically by the driver on first connect. After the user authorizes the application in their browser, **tokens are securely stored and refreshed automatically** — your application code never touches an OAuth token directly.

The driver includes **NexCore Labs' own embedded Intuit OAuth2 credentials** by default (**UseEmbeddedCredentials = True**), so most developers can connect to QuickBooks Online without registering their own Intuit Developer app. Registering your own Intuit app is only needed if you want to use your own credentials instead.

Full licenses are offered on a **tiered monthly subscription based on API read volume** — see Section 10, Licensing, for current pricing tiers.

2. Requirements

Development Environment

- Embarcadero Delphi, C++ Builder, or RAD Studio
- Delphi / C++ Builder compiler version 13 Florence (BDS 37.0), or later
- FireDAC component library, included with RAD Studio Enterprise and Architect versions
- Currently available for Win32 only; 64-bit (Win64) support is planned for a future release

QuickBooks Online

- An active Intuit QuickBooks Online account (Simple Start, Essentials, Plus, or Advanced)
- (Optional) An Intuit Developer account at developer.intuit.com and a registered Intuit app with OAuth2 credentials — only needed if you set UseEmbeddedCredentials = False
- A developer account is not required for use of the component.

NexCore Labs License

- A valid license key (trial or full)
- For the trial license, a free Intuit Developer account is required — signing up automatically provisions a Sandbox company for you to evaluate against. This takes about a minute and does not require creating an app or registering credentials. See Section 3, Step 1
- Internet access for OAuth2 authentication and license validation
- The NexCore Labs token service must be reachable from the machine running your application

Runtime Dependencies

- NxFireDACQboDriverLinkRT370.bpl — must be on the system PATH or in the application folder
- Embarcadero runtime packages — rtl370.bpl, firedac370.bpl and related FireDAC BPLs
- The driver is framework-neutral — no VCL or FMX dependency, so it works in any application type
- No additional third-party libraries required

3. Intuit Developer Setup

By default the NxQBO Driver uses NexCore Labs' own embedded Intuit OAuth2 credentials (**UseEmbeddedCredentials = True**). If you are evaluating with a trial license, Step 1 below is still required to obtain a Sandbox company — see the note below. Full-license users on embedded credentials can skip straight to Section 4, Installation, once they have their RealmID. Steps 2–4 are only needed if you want to use your own Intuit Developer app instead of NexCore Labs' embedded credentials.

If you are using the embedded credentials, leave ClientID and ClientSecret blank in the connection editor and continue to Step 5 below to find your RealmID, which is always required regardless of which credentials you use.

To use your own Intuit app, set **UseEmbeddedCredentials = False** and follow the steps below. This is a one-time setup that takes approximately 10 minutes.

Trial users: *Step 1 below is required — signing up for a free Intuit Developer account automatically gives you a Sandbox company to connect to. You do not need Steps 2–4; those are only for developers supplying their own Intuit app credentials instead of NexCore Labs' embedded ones.*

Full-license users on embedded credentials: *this entire section is optional, exactly as described below.*

Full-license users supplying their own Intuit app: *complete all five steps.*

Step 1 — Create an Intuit Developer Account

- Go to https://developer.intuit.com
- Click Sign Up and create a free developer account, or sign in with your existing Intuit account
- Once logged in you will see the Intuit Developer Dashboard

Step 2 — Create an App

- From the Dashboard click + Create an app
- Select QuickBooks Online and Payments
- Give your app a name
- Select the scopes your application needs — for the NxQBO Driver select com.intuit.quickbooks.accounting
- Click Create app

Step 3 — Get Your ClientID and ClientSecret

- In your app dashboard click Keys & credentials
- You will see two sets of keys — Sandbox and Production
- Copy the Client ID and Client Secret for the environment you are targeting
- These values go into the ClientID and ClientSecret properties of your TFDConnection

Step 4 — Set the OAuth2 Redirect URI

- In your app dashboard click Settings
- Click the Redirect URIs tab
- Click Add URI and enter: https://nxcorelabs.com/oauth/callback
- Click Save — this URL is the endpoint the NxQBO Driver uses to receive the OAuth2 code from Intuit

Step 5 — Find Your RealmID

- *For trial evaluation:*
  - Sign in at https://developer.intuit.com and go to your Sandboxes list
  - Select your Sandbox company — its RealmID is shown there

For a production connection:

- Log into https://quickbooks.intuit.com
- Look at the URL in your browser — it contains your RealmID as a numeric value after realmId=
- Alternatively go to Settings → Account and Settings → Billing & Subscription

> *Important: Keep your Client Secret secure. Do not commit it to public source control repositories.*

4. Installation

Step 1 — Run the Installer

Download the NxQBO Driver installer from https://www.nxcorelabs.com. Close RAD Studio if it is open, then run the installer as Administrator. The installer will:

- Copy the runtime BPL to the Embarcadero Bpl folder
- Copy the DCP to the Embarcadero Dcp folder
- Copy the DCU files to the library path folder
- Copy the C++Builder headers (.hpp) and import libraries (.bpi/.lib), and register both with the IDE's Include and Library paths for the classic and Clang-enhanced Win32 compilers
- Register the library path in the RAD Studio registry
- Register the design-time package in Known Packages

Step 2 — Verify Installation

- Open RAD Studio
- Watch the IDE splash screen — you should see the NexCore Labs QBO Driver listed with version 1.0.2
- Open or create a VCL project, or a project with a DataModule
- Look for the NexCore Labs tab on the component palette
- You should see TNxPhysQBODriverLink on that tab

Step 3 — Obtain a License Key

Visit https://www.nxcorelabs.com to purchase a full license or request a trial key. Your license key will be emailed to you in this format:

```
NXQBO-XXXXX-XXXXX-XXXXX-XXXXX-XXXXX-XXXXX
```

5. Quick Start — The Demo Project

Step 1 — Drop the Components

Create a new VCL Forms Application and drop these components:

`TNxPhysQBODriverLink` `TFDConnection` `TFDQuery` `TDataSource` `TDBGrid` `TDBNavigator`

Step 2 — Wire the Components

Property	Set To
FDConnection1 → Driver	QBO
TFDQuery.Connection	FDConnection1
TDataSource.DataSet	FDQuery1
TDBGrid.DataSource	DataSource1
TDBNavigator.DataSource	DataSource1

Step 3 — Configure the Connection

Double-click FDConnection1 to open the FireDAC Connection Editor. Set Driver ID to QBO, then fill in the following parameters:

Parameter	Value / Notes
UseEmbeddedCredentials	True (default) to use NexCore Labs' built-in Intuit credentials, or False to supply your own
ClientID	Your Intuit app Client ID — only needed if UseEmbeddedCredentials is False
ClientSecret	Your Intuit app Client Secret (masked in editor) — only needed if UseEmbeddedCredentials is False
RealmID	Your QBO Company ID — always required
LicenseKey	Your NexCore Labs license key
Sandbox	True for development · False for production
MinorVersion	75 (default — do not change unless advised)
LogPath	Path for log file (can be blank)
TokenPath	Leave blank to use %LOCALAPPDATA%\NxCore\QuickBooksOnline\

Step 4 — Connect and Query

```
procedure TForm1.BtnConnectClick(Sender: TObject);
begin
     FDConnection1.Connected := True;
end;

     // Open a query
FDQuery1.SQL.Text := 'SELECT * FROM Customer';
FDQuery1.Open;
```

On first connect a browser window opens asking you to log into QuickBooks Online and authorize your application. After authorization the browser closes, tokens are stored securely, and the connection is established. On all subsequent connects the stored tokens are used automatically — no browser interaction is needed.

Step 5 — Reading Data

```
// Navigate records
FDQuery1.First;
while not FDQuery1.Eof do
begin
  MemoLog.Lines.Add(FDQuery1.FieldByName('DisplayName').AsString);
  FDQuery1.Next;
end;

// Locate a record
FDQuery1.Locate('DisplayName', 'Acme Corp', [loCaseInsensitive]);

// Apply an in-memory filter
FDQuery1.Filter := 'Balance > 1000';
FDQuery1.Filtered := True;
```

Step 6 — Writing Data

```
// Insert a new Customer
FDQuery1.SQL.Text :=
  'INSERT INTO Customer (DisplayName, CompanyName) ' +
  'VALUES (:DisplayName, :CompanyName)';
FDQuery1.ParamByName('DisplayName').AsString := 'Acme Corp';
FDQuery1.ParamByName('CompanyName').AsString := 'Acme Corporation';
FDQuery1.ExecSQL;

// Update a Customer — Id and SyncToken are required by QBO,
// in the WHERE clause only — never in SET
FDQuery1.SQL.Text :=
  'UPDATE Customer SET DisplayName = :DisplayName ' +
  'WHERE Id = :Id AND SyncToken = :SyncToken';
FDQuery1.ParamByName('DisplayName').AsString := 'Acme Corp (Updated)';
FDQuery1.ParamByName('Id').AsString := '123';
FDQuery1.ParamByName('SyncToken').AsString := '0';
FDQuery1.ExecSQL;
```

QBO requires Id and SyncToken for all UPDATE and DELETE operations. These fields are returned in every SELECT result and must be preserved by your application if you intend to update or delete a record later. SyncToken belongs only in the WHERE clause — never in SET.

Supported QBO Entities

Account Bill BillPayment Budget Class CompanyInfo CreditMemo Customer Department Deposit Employee Estimate Invoice Item JournalEntry Payment PaymentMethod Purchase PurchaseOrder RecurringTransaction RefundReceipt SalesReceipt TaxAgency TaxCode TaxRate Term TimeActivity Transfer Vendor VendorCredit

6. Supported Operations

This section describes which standard FireDAC dataset operations the NxQBO driver supports, and how QuickBooks Online's design affects certain operations. QuickBooks Online is a stateless REST API rather than a traditional SQL database; the driver maps FireDAC's data-access model onto it.

Reading Data (SELECT)

Fully supported through TFDQuery: SELECT with column lists or SELECT *, WHERE with =, !=, <, >, <=, >=, LIKE, AND, and IN, comparison on text, dates, and numeric values, ORDERBY on single or multiple fields with ASC/DESC, and pagination via MAXRESULTS and STARTPOSITION.

QuickBooks Online requires all WHERE comparison values to be single-quoted, including numbers. The driver handles this automatically — you write standard SQL and the driver formats it correctly. For example, `WHERE Balance > 0` is sent to QuickBooks as `WHERE Balance > '0'`.

> *Note on the Id field: QuickBooks Online supports only = and IN on the Id field. The operators >, <, >=, <=, !=, and LIKE are not valid on Id.*

Dataset Navigation and In-Memory Operations

Fully supported. Once a result set is open, common dataset operations work normally because they operate on the in-memory rowset.

First Next Prior Last MoveBy RecordCount RecNo Bof Eof Locate Lookup Filter Filtered IndexFieldNames SaveToFile SaveToStream LoadFromFile LoadFromStream TFDMemTable

To use SaveToFile with the JSON format (sfJSON), add `FireDAC.Stan.StorageJSON` to your uses clause. The XML and binary formats require `FireDAC.Stan.StorageXML` and `FireDAC.Stan.StorageBin` respectively.

Creating Records (INSERT)

Supported via explicit INSERT statements through ExecSQL. The driver returns the new record's Id and SyncToken where applicable after a successful insert. For entities with nested structures, supply the nested portion as a JSON value in the parameter.

Updating Records (UPDATE)

Supported via explicit UPDATE statements through ExecSQL. QuickBooks Online requires the current SyncToken on every update and performs a sparse update — only the fields you specify are changed. Always read the record first to obtain its current SyncToken.

Deleting Records (DELETE)

Behavior depends on the QuickBooks Online entity type. Transaction entities support a true hard delete — both Id and SyncToken are required, and the record is permanently removed from QuickBooks Online. Name-list entities cannot be hard-deleted; instead they are deactivated with a sparse update setting Active to false.

Live-Bound Editing (Edit / Post)

Fully supported. In addition to explicit INSERT/UPDATE/DELETE statements, the NxQBO Driver supports the standard FireDAC live-dataset pattern — Edit, modify a field, Post — and writes the change straight through to QuickBooks Online. This means grid-bound editing works out of the box.

```
FDQuery1.SQL.Text := 'SELECT * FROM Customer WHERE Id = ''123''';
FDQuery1.UpdateOptions.KeyFields := 'Id;SyncToken';
FDQuery1.UpdateOptions.UpdateMode := upWhereKeyOnly;
FDQuery1.UpdateOptions.UpdateTableName := 'Customer';
FDQuery1.Open;

// Edit a field - Post sends a sparse update with just the
// changed field(s) plus Id and SyncToken
FDQuery1.Edit;
FDQuery1.FieldByName('DisplayName').AsString := 'Acme Corp (Updated)';
FDQuery1.Post;

// A second edit works immediately - no re-open needed
FDQuery1.Edit;
FDQuery1.FieldByName('CompanyName').AsString := 'Acme Corporation Inc.';
FDQuery1.Post;

// Deactivate a name-list entity via Edit/Post
FDQuery1.Edit;
FDQuery1.FieldByName('Active').AsBoolean := False;
FDQuery1.Post;

// Nested/array fields - assign JSON directly
FDQuery1.Edit;
FDQuery1.FieldByName('Line').AsString :=
  '[{"Amount":250.00,"DetailType":"SalesItemLineDetail",' +
  '"SalesItemLineDetail":{"ItemRef":{"value":"1"}}}]';
FDQuery1.Post;
```

After a successful Post, the driver automatically refreshes the row's SyncToken from QBO's response, so the dataset stays valid for a second edit without requiring you to re-open or re-query. Edit/Post uses the same underlying write path as explicit UPDATE statements.

Unsupported SQL

QuickBooks Online is a stateless REST API, not a traditional SQL database — so a few standard SQL capabilities don't apply here:

- Stored procedures
- Server-side scrollable cursors
- Traditional transactions, commit, rollback, and savepoints
- Joins across entities in a single query

Working around this: query each entity separately, then relate the results in your application code — this is the normal pattern for REST-backed FireDAC drivers, not a workaround specific to QBO.

7. More Examples

INSERT (Create)

Customer

```
FDQuery1.SQL.Text := 'INSERT INTO Customer (DisplayName, CompanyName, ' +
  'PrimaryEmailAddr, PrimaryPhone) ' +
  'VALUES (:DisplayName, :CompanyName, ' +
  ':PrimaryEmailAddr, :PrimaryPhone)';
FDQuery1.ParamByName('DisplayName').AsString := 'Acme Corporation';
FDQuery1.ParamByName('CompanyName').AsString := 'Acme Corporation';
FDQuery1.ParamByName('PrimaryEmailAddr').AsString := '{"Address":"info@acme.com"}';
FDQuery1.ParamByName('PrimaryPhone').AsString := '{"FreeFormNumber":"555-123-4567"}';
FDQuery1.ExecSQL;
LNewId := GNxQBOLastInsertedId;
```

Invoice

```
FDQuery1.SQL.Text := 'INSERT INTO Invoice (CustomerRef, Line, TxnDate) ' +
  'VALUES (:CustomerRef, :Line, :TxnDate)';
FDQuery1.ParamByName('CustomerRef').AsString :=
  '{"value":"123","name":"Acme Corporation"}';
FDQuery1.ParamByName('Line').AsString :=
  '[{"DetailType":"SalesItemLineDetail",' +
  '"Amount":250.00,' +
  '"SalesItemLineDetail":{"ItemRef":{"value":"1","name":"Services"}}}]';
FDQuery1.ParamByName('TxnDate').AsString := '2026-06-16';
FDQuery1.ExecSQL;
LNewId := GNxQBOLastInsertedId;
```

Update Invoice — Change Amount

```
{ Step 1 – fetch current SyncToken }
FDQuery1.SQL.Text := 'SELECT Id, SyncToken FROM Invoice WHERE Id = :Id';
FDQuery1.ParamByName('Id').AsString := '456';
FDQuery1.Open;
LId := FDQuery1.FieldByName('Id').AsString;
LSyncToken := FDQuery1.FieldByName('SyncToken').AsString;
FDQuery1.Close;

{ Step 2 – update }
FDQuery1.SQL.Text := 'UPDATE Invoice SET Line = :Line ' +
  'WHERE Id = :Id AND SyncToken = :SyncToken';
FDQuery1.ParamByName('Line').AsString :=
  '[{"DetailType":"SalesItemLineDetail",' +
  '"Amount":500.00,' +
  '"SalesItemLineDetail":{"ItemRef":{"value":"1","name":"Services"}}}]';
FDQuery1.ParamByName('Id').AsString := LId;
FDQuery1.ParamByName('SyncToken').AsString := LSyncToken;
FDQuery1.ExecSQL;
```

Delete an Invoice (Hard Delete — Transaction Entity)

```
{ Step 1 – fetch current SyncToken }
FDQuery1.SQL.Text := 'SELECT Id, SyncToken FROM Invoice WHERE Id = :Id';
FDQuery1.ParamByName('Id').AsString := '456';
FDQuery1.Open;
LId := FDQuery1.FieldByName('Id').AsString;
LSyncToken := FDQuery1.FieldByName('SyncToken').AsString;
FDQuery1.Close;

{ Step 2 – hard delete }
FDQuery1.SQL.Text := 'DELETE FROM Invoice ' +
  'WHERE Id = :Id AND SyncToken = :SyncToken';
FDQuery1.ParamByName('Id').AsString := LId;
FDQuery1.ParamByName('SyncToken').AsString := LSyncToken;
FDQuery1.ExecSQL;
```

Deactivate a Customer (When Delete Is Not Permitted)

```
{ Customers with transactions cannot be hard-deleted – deactivate instead }
FDQuery1.SQL.Text := 'SELECT Id, SyncToken FROM Customer WHERE Id = :Id';
FDQuery1.ParamByName('Id').AsString := '123';
FDQuery1.Open;
LId := FDQuery1.FieldByName('Id').AsString;
LSyncToken := FDQuery1.FieldByName('SyncToken').AsString;
FDQuery1.Close;

FDQuery1.SQL.Text := 'UPDATE Customer SET Active = :Active ' +
  'WHERE Id = :Id AND SyncToken = :SyncToken';
FDQuery1.ParamByName('Active').AsBoolean := False;
FDQuery1.ParamByName('Id').AsString := LId;
FDQuery1.ParamByName('SyncToken').AsString := LSyncToken;
FDQuery1.ExecSQL;
```

DELETE — Entity-Specific Behavior: Transaction entities are permanently hard-deleted. Name-list entities are deactivated instead. Always use Id and SyncToken as strings, and always fetch the current SyncToken immediately before writing.

Key Rules — Summary for the Guide

Rule	Detail
Always use params	Never put values as literals in SQL — the driver reads params to build the QBO JSON body
SyncToken in WHERE only	Never put SyncToken in the SET list — QBO manages it server-side
Fetch SyncToken first	Always SELECT immediately before UPDATE, DELETE, or Edit/Post — a stale SyncToken causes a concurrency rejection
Id and SyncToken are strings	Always use AsString — QBO rejects them if serialized as integers
Nested fields use JSON strings	PrimaryEmailAddr, CustomerRef, Line, etc. are JSON objects/arrays passed as string params, or assigned directly via Edit/Post
GNxQBOLastInsertedId	Read this global after ExecSQL on an INSERT to get the QBO-assigned Id
DELETE is entity-specific	Transaction entities are hard-deleted permanently; name-list entities are deactivated instead
Edit/Post refreshes SyncToken	After a successful Post, the driver updates the row's SyncToken automatically

8. Error Handling and Display

The NxQBO driver is framework-neutral — it has no dependency on the VCL or FireMonkey (FMX) libraries. This means the same driver works in VCL applications, FMX multi-device applications, console applications, Windows services, and web or application servers, all from a single package.

Because the driver cannot assume a user interface is present, it never displays a dialog on its own by default. Instead it logs errors when a LogPath is configured and exposes an optional callback hook that your application can assign.

Default Behavior (Log Only)

```
GNxQBOShowDialogs : Boolean;        // default False - no dialog, log only
GNxQBOOnError     : TNxQBOErrorEvent; // default nil - your display hook
```

Option 1 — Enable the Built-In Windows Dialog

```
uses NxQBOErrorHandler;

// e.g. in your form's OnCreate or app startup
GNxQBOShowDialogs := True;
```

Option 2 — Route Errors to Your Own Display (Recommended)

```
uses NxQBOErrorHandler, Vcl.Forms;

procedure TForm1.HandleQBOError(const AMessage, ACode, ADetail: String);
begin
  Application.MessageBox(PChar(AMessage), 'QuickBooks Online',
    MB_OK or MB_ICONERROR);
end;

// assign during startup:
GNxQBOOnError := HandleQBOError;
```

Servers and Services

For a Windows service, a web/application server, or any headless process, leave both globals at their defaults. The driver will log errors and never attempt to show a dialog.

> *Tip: Regardless of which option you choose, always set the LogPath connection parameter in production so errors are recorded to disk for diagnosis.*

9. Property Reference

All properties are set in the FireDAC Connection Editor or programmatically via TFDConnection.Params.

Property	Type	Req	Default	Description
UseEmbeddedCredentials	Boolean	No	True	Use NexCore Labs' embedded Intuit credentials. Set False to supply your own ClientID/ClientSecret.
ClientID	String	If not embedded	None	Your Intuit app Client ID. Only needed if UseEmbeddedCredentials is False.
ClientSecret	String	If not embedded	None	Your Intuit app Client Secret. Masked in the editor.
†RealmID	String	Yes	None	The QuickBooks Online Company ID. The ID of the QuickBooks Online company to connect to. **Leave blank for a customer's first connection** — the driver discovers the correct RealmID automatically from Intuit's own OAuth consent screen once the user signs in and selects a company. Only set this explicitly if you need to target a specific company programmatically for a user with access to more than one, or during sandbox/development testing.
LicenseKey	String	Yes	None	Your NexCore Labs license key. Validated against the license server on every connect.
Sandbox	Boolean	No	False	True connects to the Intuit QBO Sandbox environment. False for production.
MinorVersion	Integer	No	75	The QBO API minor version number. Do not change unless advised by support.
LogPath	String	No	None	Full path to the driver log file. Blank disables logging. The driver creates the folder if needed.
TokenPath	String	No	None	Folder for OAuth2 token cache files. Blank uses %LOCALAPPDATA% \NxCore\QuickBooksOnline\.

Note: TFDConnection.Ping always returns True for this driver and does not verify connectivity — use a real query to confirm the connection is live.

Note: †You do not need to look up or enter a RealmID before your customer's first connection. Just leave it blank — Intuit's sign-in screen handles company selection, and the driver fills in the correct value automatically once the user authorizes.

10. Licensing

Trial License

A free trial license is available for evaluation. During the trial period, all operations are fully functional against a QuickBooks Online Sandbox company. Trial licenses do not connect to production QuickBooks Online — a Full License is required to connect to a production company.

To request a trial license visit https://www.nxcorelabs.com.

Full License — Subscription Tiers

Full licenses are billed monthly based on your API read volume. All tiers include the NexCore Labs OAuth2 token service, license validation, component updates, new QBO entity support, and technical support.

Tier	Monthly Price	API Reads Included
Standard	$25/month	15,000 reads
Plus	$39/month	25,000 reads
Pro	$69/month	50,000 reads
Enterprise	Contact us	Custom volume

If you're unsure which tier fits your application, start with Standard — you can upgrade at any time as your usage grows. Contact support@nxcorelabs.com for Enterprise pricing or if you need volume beyond the Pro tier.

License Terms

Each license key is tied to a single activated machine. If you need to connect to multiple QBO companies, a separate license is required for each. Contact NexCore Labs for multi-company or multi-seat pricing.

You may request to have your license moved to another machine up to five (5) times, using the contact form at https://nxcorelabs.com/contact and choosing "Move License." If you need to move the license more than five times, please contact us using the same contact form.

11. Support

Email: support@nxcorelabs.com

Website: https://www.nxcorelabs.com/contact

When submitting an issue please include:

- Your RAD Studio version
- The NxQBO Driver version (shown in the IDE splash screen)
- The contents of your log file (if using a log file)
- A description of the steps to reproduce the issue

12. API Published Properties

The following properties are published and intended for use by developers. They are accessible via the Object Inspector, the FireDAC Connection Editor, or programmatically through TFDConnection.Params and the underlying connection and command objects.

Connection Definition Parameters

Set via the FireDAC Connection Editor or TFDConnection.Params for a connection with DriverName = QBO.

Property	Type	Access	Description
UseEmbeddedCredentials	Boolean	Read/Write	True (default) uses NexCore Labs' embedded Intuit credentials. False requires your own ClientID/ClientSecret.
ClientID	String	Read/Write	QBO OAuth2 Client ID from your Intuit Developer app. Only used when UseEmbeddedCredentials is False.
ClientSecret	String	Read/Write	QBO OAuth2 Client Secret. Masked in the Connection Editor. Only used when UseEmbeddedCredentials is False.
RealmID	String	Read/Write	QuickBooks Online Company/Realm ID.
LicenseKey	String	Read/Write	NexCore Labs license key, format NXQBO-XXXXX-XXXXX-XXXXX-XXXXX-XXXXX-XXXXX.
Sandbox	Boolean	Read/Write	True connects to the QBO Sandbox environment. False uses production. Default True.
MinorVersion	Integer	Read/Write	QBO API minor version number. Leave at default unless advised by your company's version usage.
LogPath	String	Read/Write	Full path to the driver diagnostic log file. Blank disables logging.
TokenPath	String	Read/Write	Folder where OAuth2 token cache files are stored. Blank uses the default %LOCALAPPDATA%\NxCore\QuickBooksOnline\ path.

Runtime Connection Properties (TNxPhysQuickbooksOnlineConnection)

Read-only properties available on the active physical connection object after a successful connect.

Property	Type	Access	Description
AccessToken	String	Read-Only	The current OAuth2 access token in use for this connection.
RealmID	String	Read-Only	The active company/realm ID for this connection.
BaseURL	String	Read-Only	The sandbox or production base URL currently in use.
MinorVersion	Integer	Read-Only	The active QBO API minor version for this connection.
LicensePlan	String	Read-Only	Returns trial or full, reflecting the current license status.
LicenseDaysRemaining	Integer	Read-Only	Days remaining on a trial license. Returns -1 for a full license, which never expires.
LastTID	String	Read-Only	Intuit's intuit_tid response header from the most recent QBO API call — quote this when opening an Intuit support ticket. Updates on every SELECT and DML request, including failed calls.
LastResponseHeaders	TStrings	Read-Only	The full header set from the most recent QBO API response, as Name=Value lines.