APPX LIBRARY

# PostgreSQL 18 with APPX 5.5.4 on Rocky Linux

PostgreSQL 18.7 · Rocky Linux 9.8 · APPX 5.5.4 (32-bit engine)

> Set up PostgreSQL as an APPX file system using FMS type 6. Start with one file, then save a record in APPX and check it in PostgreSQL.

APPX Software, Inc.

Made on November 18, 2026 from the pages of the APPX Library, https://wiki.appx.com/manuals/library/rdbms-postgresql-18-7-rocky-9-8-appx-554-x86/. The pages are the current version.

# Contents

Chapter 1 **What to install** ..... 3

Chapter 2 **Choose a database and create a login** ..... 4

Chapter 3 **Set up the 32-bit System DSN** ..... 5

Chapter 4 **Set up the FMS group and identity** ..... 7

Chapter 5 **Assign one file and create the table** ..... 8

Chapter 6 **Save and check a record** ..... 9

Chapter 7 **Troubleshooting** ..... 11

CHAPTER 1

# What to install

This example uses the 32-bit APPX 5.5.4 engine on Rocky Linux 9.8, PostgreSQL 18.7, and 32-bit psqlODBC 18.00.0001. The APPX server uses unixODBC 2.3.12. The lab ran PostgreSQL in a container on the same Rocky Linux host. If PostgreSQL runs elsewhere, install the ODBC driver on the APPX server and use the PostgreSQL host and port in the DSN.

| Component | Download |
|---|---|
| **APPX server** | Install the 32-bit engine. On x86-64 Rocky Linux, use the installer for 64-bit Linux. It installs the 32-bit APPX engine. APPX 5.5.4 Linux server installer · Installation instructions |
| **APPX Desktop Client** | Use the 5.5.4 Desktop Client to open the APPX session. Desktop Client installer |
| **PostgreSQL** | Install PostgreSQL 18 x64. Use `SELECT version();` to verify an existing server. PostgreSQL installation |
| **psqlODBC** | Install the 32-bit PostgreSQL ANSI driver on the APPX server. Match the driver and driver manager to the APPX engine's bitness. 18.00.0001 source archive · Unix build instructions |

Install the 32-bit driver manager on the APPX server.

```
sudo dnf install unixODBC.i686
```

Build the driver against 32-bit libpq and unixODBC development files. The lab used 32-bit libpq 13.18 and unixODBC 2.3.12. Adjust the two prefixes below to the actual development installations.

```
 autoreconf -fi
CFLAGS='-m32 -O2' \
LDFLAGS='-m32 -Wl,-rpath,/usr/local/pgsql-odbc32/lib' \
  ./configure --with-libpq=/usr/local/pgsql-odbc32 \
    --with-unixodbc=/usr/local/unixodbc32 --prefix=/opt/psqlodbc32
make
sudo make install
```

CHAPTER 2

# Choose a database and create a login

Ask your DBA for the PostgreSQL host, TCP port, and database. The walkthrough uses a Windows-1252 database.

| Example database | Server encoding | Use |
| --- | --- | --- |
| **appxsb** | `WIN1252` | Single-byte setup used in this walkthrough. |

Connect to `postgres` as a DBA in `psql`. Enter a new password when `\password` prompts you. Run `CREATE DATABASE` for a new database.

```
CREATE ROLE appx_pg554_demo LOGIN;
\password appx_pg554_demo

CREATE DATABASE appxsb OWNER appx_pg554_demo
    TEMPLATE template0 LOCALE_PROVIDER libc LOCALE 'C' ENCODING 'WIN1252';

REVOKE CONNECT ON DATABASE appxsb FROM PUBLIC;
\connect appxsb
REVOKE CREATE ON SCHEMA public FROM PUBLIC;
GRANT USAGE, CREATE ON SCHEMA public TO appx_pg554_demo;
```

The login needs to connect to the database and create objects in the target schema. This example uses `public` in dedicated test databases. PostgreSQL CREATE DATABASE reference

CHAPTER 3

# Set up the 32-bit System DSN

Run `odbcinst -j` to find `odbcinst.ini` and the system `odbc.ini`. Register the driver in `odbcinst.ini` using its installed path. Run `file` on the driver to verify that it is 32-bit.

```
[PostgreSQL ANSI]
Driver=/opt/psqlodbc32/lib/psqlodbca.so
```

Use these values to configure the DSN. The standard PostgreSQL port is `5432`. The isolated lab used `55487`. Check the actual port on the server.

| DSN setting | Example |
|---|---|
| **Data Source** | `APPX-POSTGRES` |
| **Server and Port** | `127.0.0.1` and `5432` |
| **Database** | `appxsb` |
| **User Name for Test** | `appx_pg554_demo` |
| **SSL Mode** | `disable` for the isolated loopback test |

```
[APPX-POSTGRES]
Driver=PostgreSQL ANSI
Servername=127.0.0.1
Port=5432
Database=appxsb
SSLmode=disable
BoolsAsChar=0
LFConversion=0
ConnSettings=SET client_encoding TO 'WIN1252'
```

In the engine's `appx.env`, point `APPX_ODBC_LIB` to the 32-bit driver manager and `ODBCSYSINI` to the directory containing the two INI files. This example uses the system files.

```
APPX_ODBC_LIB=/usr/lib/libodbc.so.2
ODBCSYSINI=/etc
```

Test from the APPX engine's account using the same `ODBCSYSINI` value. Use a 32-bit `isql`. The lab built it under `/usr/local/unixodbc32`. The command prompts for the test password.

```
read -rsp 'PostgreSQL test password: ' pg_test_password; printf '\n'
ODBCSYSINI=/etc /usr/local/unixodbc32/bin/isql -v APPX-POSTGRES appx_pg554_demo "$pg_test_password"
unset pg_test_password
```

**Can't connect to server (or driver error).** If `isql` connects but APPX shows this error, compare its `ODBCSYSINI` with the directory used for `isql`. Correct the path in `appx.env` and start a fresh APPX engine session.

Use `verify-full` and a certificate trusted by the APPX server for a remote PostgreSQL server. PostgreSQL TLS options

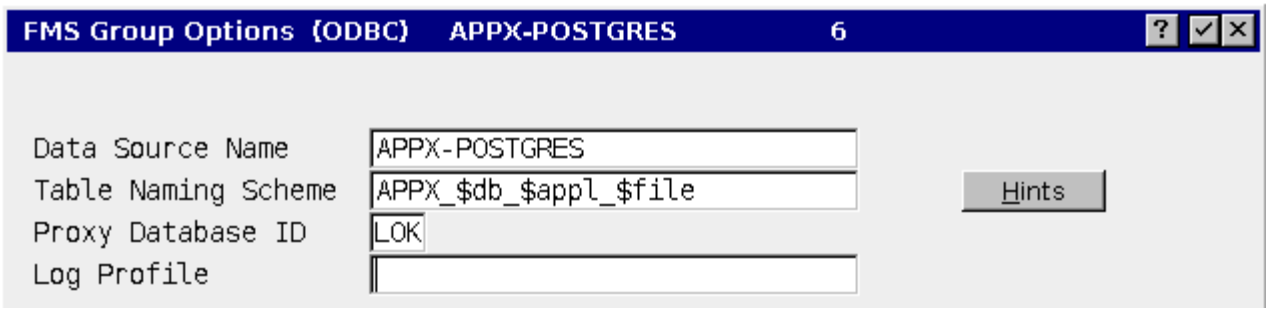
CHAPTER 4

# Set up the FMS group and identity

Open a new APPX Desktop Client session. Choose **System Administration → Configuration → File System Groups** under **File Maintenance**.

1. Add `APPX-POSTGRES` and enter FMS type `6`.
2. Click **FMS Group Attributes.** Enter these values and save. Reserve an unused three-character ID for proxy locks. This example uses `LOK`.

| FMS option | Value |
|---|---|
| Data Source Name | `APPX-POSTGRES` |
| Table Naming Scheme | `APPX_$db_$appl_$file` |
| Proxy Database ID | `LOK` |




ODBC FMS options for the PostgreSQL System DSN.

On the group row, click **Identities.** Enter your APPX user ID to give that user a database login. Leave **APPX User** blank to set a default identity for the group. Press Enter to accept the key and again to open the details.

Enter `appx_pg554_demo` in **User Id** and leave **Realm** blank. Save the identity and reopen it in **Change** mode. Click **Change Password**, enter the database password, and press `Tab` to move to **Confirm Password**. Enter it again and click **Change Password**. When APPX reports **Password Changed, END to Exit**, close the password dialog and save the identity. Open a fresh APPX session before testing.

**You Must Provide A Proxy Database For This File.** Enter the **Proxy Database ID**, save the FMS group, and retry in a fresh session. APPX uses its proxy file to coordinate record holds among APPX sessions. Plan separately for direct SQL writers. APPX ODBC file system settings

CHAPTER 5

# Assign one file and create the table

Use a test database and a file you can create empty. This example uses the Demo Application's `COUNTRY` file.

1. Open **System Administration → Database/Applications → Databases / Applications** under **File Maintenance**. Add `TST` with the company name **PostgreSQL Test**. Set the startup application to `DMO`, version `00`. Save and accept the report defaults.
2. For `TST`, open **Related Applications**. Add `DMO` in **Live Operations** if it is absent. Select the `DMO` row and click **Database Management**. If APPX prompts for a **Language ID**, set `ENG` and reopen **Database Management**.
3. In **File Selection**, click **Unselect All**. Highlight `COUNTRY` and press `F3`. If you use the **Select** button, enter `COUNTRY` in **Name** and confirm. Return to **Data File Management** and confirm that one file is selected.
4. Open **File Specifications**. Set **FMS** to `6` and **FMS Group** to `APPX-POSTGRES` for `COUNTRY`. Save and return.
5. Choose **Create Files**. The result for `COUNTRY` should say **Created**.

File Specifications

| DB | Ap | File | Records | FMS | FMS Group | Use DB | --Opts-- FA OP FO |
|---|---|---|---|---|---|---|---|
| TST | DMO | COUNTRY | 100 | 6 | APPX-POSTGRES | | |

File Specifications for COUNTRY in the scratch database.

Create Files

| DB | Ap | Ver | File | Status/Message |
|---|---|---|---|---|
| TST | DMO | 00 | COUNTRY | Created |

Create Files reports Created for COUNTRY.

The naming scheme expands to `APPX_TST_DMO_COUNTRY`. PostgreSQL folds the unquoted name to lowercase. In this example the table is `public.appx_tst_dmo_country`.

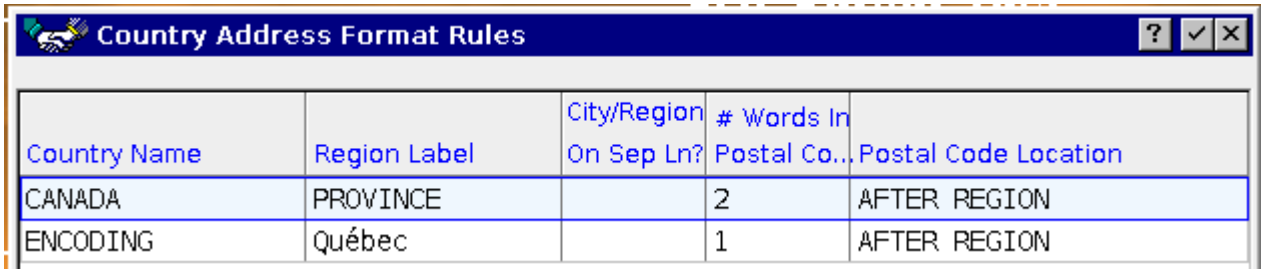
For an existing APPX/IO file, preserve an APPX/IO backup and rehearse the conversion on a copy. APPX/IO migration and rollback guide

CHAPTER 6

# Save and check a record

1. From the APPX main menu, choose **Run an Application** and execute database `TST`. In the demo's top menu, choose **Setup → Countries**.
2. In **Country Address Format Rules**, choose **Add**. Enter `CANADA` for **Country Name**, `PROVINCE` for **Region Label**, and `2` for **# Words In Postal Code**. Leave **Postal Code Location** at **AFTER REGION**. Click **Save**.
3. Repeat with `ENCODING` for **Country Name**, `Québec` for **Region Label**, and `1` for **# Words In Postal Code**. Save.
4. Open a fresh Desktop Client session, run `TST`, and go back to **Setup → Countries**. Choose **Inquire** and click **First Record** if the key field is empty. Find the two keys and confirm the saved values, including the accent in `Québec`.

**Cannot Access Required File for PROSPECT.** In a test database with only `COUNTRY` created, click **Cancel** on this message to continue the Countries test. Create the full demo database to run the rest of the application.




The saved records in a fresh Desktop Client session.

In `psql`, connect as `appx_pg554_demo` and select `appxsb`. Check the two keys and the unique index.

```
psql -h 127.0.0.1 -p 5432 -U appx_pg554_demo -W -d appxsb
```

```
\encoding UTF8
SHOW server_encoding;
SELECT btrim(country_name) AS country_name,
       btrim(country_region_label) AS region_label, country_no_words,
       encode(convert_to(btrim(country_region_label),
                         getdatabaseencoding()::name), 'hex') AS server_hex
FROM public.appx_tst_dmo_country
WHERE country_name IN ('CANADA', 'ENCODING')
ORDER BY country_name;

SELECT indexdef FROM pg_indexes
WHERE schemaname = 'public' AND tablename = 'appx_tst_dmo_country';
```

Expect `CANADA / PROVINCE / 2` and `ENCODING / Québec / 1`. The Windows-1252 bytes for `Québec` are `5175e9626563`. The unique index should include `country_name`.

CHAPTER 7

# Troubleshooting

## Check the APPX engine

This guide uses **FMS type 6 (ODBC)** through the PostgreSQL ODBC driver.

In APPX **Release Information**, record **Engine Release** and **Compile Date**. The application release numbers on the same screen identify the APPX system applications. Match the database client or ODBC driver to the bitness of the engine that runs the operation. Check the settings on the APPX server and under the Windows or Linux account that runs the engine.

## Test the connection outside APPX

On the APPX server, use the 32-bit `isql` and the DSN from **FMS Group Attributes**. Check the driver, server, port, and database. Use the same login and password as APPX. Run `odbcinst -j` from the matching unixODBC installation to see its configuration paths. Check `ODBCSYSINI` and `ODBCINI` in the engine's environment. If the driver fails to load, check its bitness with `file` and its dependencies with `ldd`.

For a timeout or connection refusal, check the database service, listening address and port, and firewall. A database authentication error means the request reached the server. Check the matching server log entry for the reason it rejected the login.

A `no pg_hba.conf entry` message points to the client address, database, user, or connection type in the authentication rules. A `password authentication failed` message means PostgreSQL selected an authentication rule and rejected the login. Look for the matching `FATAL` and `DETAIL` lines in the PostgreSQL server log. PostgreSQL authentication messages

## Check the FMS group and Identity

If APPX reports **Can't connect to server (or driver error)** after a successful DSN test, confirm that the FMS group's **Data Source Name** matches the tested System DSN. The DSN test uses the credentials supplied to the test. APPX uses the credentials from its FMS Identity.

Under **Identities**, look for an active, unexpired entry for the APPX user running the operation. That entry takes precedence over the blank default. Check **User Id** and re-enter the database password with **Change Password**. Save the Identity and open a fresh APPX session. APPX can cache connection and Identity settings for the life of the session.

If the same password works in the DSN test and fails in APPX, check the driver's connection string rules and the password length supported by the APPX release. A dedicated test login with a newly generated alphanumeric password can help isolate a parsing or length problem.

### Check where APPX creates the table

If the connection succeeds, read the **Create Files** result or open **View Log** for the database error. Check the database, login, and target schema or database with the same connection settings as APPX. Compare the APPX database ID, application ID, file name, and **Table Naming Scheme** with the SQL table name.

The DSN's **Database** names the database. PostgreSQL creates an unqualified table in the current schema. A `permission denied for schema public` error means the statement reached `public`. If the DBA assigned a schema such as `appx_data`, select that schema for the APPX connection.

In the PostgreSQL DSN's section of `odbc.ini`, set `ConnSettings=SET search_path TO appx_data`.

Keep the DSN's database name. Use a separate DSN for APPX if other applications use the existing DSN. If a grant is missing, ask the DBA to connect to the target database and grant `USAGE, CREATE` on the assigned schema to the APPX login. Apply the setup example's `REVOKE` commands only when preparing a new database you administer.

```
SELECT current_database(), current_user, current_schema();
SELECT has_schema_privilege(current_user, 'appx_data', 'USAGE') AS can_use,
       has_schema_privilege(current_user, 'appx_data', 'CREATE') AS can_create;
SELECT to_regclass('appx_data.appx_tst_dmo_country');
```

Run the first query through the same DSN to verify the connection's search path. Substitute your schema and table in these queries. After **Create Files** reports **Created**, confirm the table and its key index in the assigned schema. PostgreSQL schema search path · psqlODBC Connect Settings

For a proxy file error, check the FMS group's **Proxy Database ID** (`LOK` in this example) and the engine account's access to the proxy files under `APPXPATH`. If APPX reports that a record is held, check whether another APPX session has it open in Change mode.

### Capture one failing operation

1. Make a copy of the `appx.env` under the effective `APPXPATH` of the engine that runs the process. Create a private log directory that the engine account can write to, then temporarily set the variables below. Keep `$(pid)` as written. The SQL logger replaces it with the engine process ID.

   ```
   APPX_SQL_CMD=/var/tmp/appx-trace/appx-rdbms-$(pid).log
   APPX_DBG_CODE=0x201F
   ```

   The mask includes generated SQL, return statuses, fetch attempts, and engine and connection operations, plus proxy lock operations where applicable. Use `0xFFFFFFFF` for one attempt if support needs more detail. The engine reads these settings on startup. `APPX_ODBC_LOG` and `APPX_ODBC_CODE` apply to the driver used by external programs to read APPX data.

2. If the APPX SQL log lacks the driver error, add `Trace=Yes` and `TraceFile=/var/tmp/appx-trace/odbc.log` under `[ODBC]` in the engine's `odbcinst.ini`. This can trace other applications using that configuration. Limit the capture to one attempt. unixODBC tracing settings
3. Open a fresh APPX session and reproduce the failure once. For **Create Files**, select only the intended test file. For a failed `READ` with `HOLD`, note the APPX error and flags before the next `WRITE`. Try a plain `READ` of the same key in another fresh session. If the log is missing, check `APPXPATH`, the effective variables, and the engine account's access to the log directory.
4. Exit the test sessions and restore the previous `appx.env` settings. If you enabled unixODBC tracing, restore the previous `odbcinst.ini` settings.

If an insert follows a failed held read, keep the first error. A later duplicate-key error indicates that the key existed when the insert ran. Use the insert branch only when the read explicitly reports that the record was absent. Handle lock and database errors before trying another write.

In the ODBC trace, find the first failed call and its diagnostic records. For `SQLDriverConnect` (which may appear as `SQLDriverConnectW`), capture the return code, SQLSTATE, and full driver message. `IM002` points to DSN or driver lookup. `08001` covers several connection failures, so read the message that accompanies it.

Share the SQL statement, fetch or lock result, and the first SQLSTATE, native error number, and message when available. Include the APPX flags, file and operation, timestamp and time zone, APPX engine release, build date and bitness, operating system, driver name and version, and database versions before and after an upgrade. Keep the SQL structure and error codes intact and replace passwords and record values with placeholders. The APPX SQL log can contain the engine environment and full connection strings. APPX environment settings

If the query returns no row after an APPX save, confirm `current_database()` in SQL and the **Data Source Name** in APPX. Open a fresh APPX session after changing the DSN.

For SQLSTATE `42501` during **Create Files**, check that the login has `USAGE` and `CREATE` on the schema shown in the PostgreSQL error.