Practical twinBASIC:
Use Cases for Access Developers

twinBASIC is a project by Wayne Phillips

creator of vbWatchdog and other fine products at everythingaccess.com

Presented by Mike Wolfe of nolongerset.com

What is twinBASIC?

- Fully Compiled Language
- 100% Backwards Compatible (VBA/VB6)
- New Language Features
  - Generics, class constructors, Return syntax, Unicode, inline initialization, AndAlso/OrElse, method overloading, full OOP

A Complement to Access

- Make EXEs, DLLs, ActiveX, Win Services
- Familiar VBA syntax
- Multi-threaded
- Zero dependencies (no Java Runtime, no .NET framework)
- Static Linking (Can embed SQLite, Compression libraries)

2026 Update

- **Fusion Tech**: 32-bit ActiveX in 64-bit
- **Full OOP**: Mult. inheritance, overrides
- **Cmd line compile**: 32/64 bit flags
- **Long Tail**: VBx compatibility fixes
- **New legal entity**: Prep for v1 Launch

Folder Watcher: Why bother?

	VBA (Form Timer)	FolderWatcher (twinBASIC)
Response time	Seconds (timer interval)	Instant (OS-level event)
Side effects	Increased bug surface	Fewer moving parts
Filename source	Loop with Dir()	From the OS event

⚠ *VBA Gotchas: Dir() keeps global state, form timer strips trailing spaces, DoEvents race conditions*

Demo:
Folder Watcher

Watch a Folder for Incoming Files

How Folder Watcher Works

Directory Handle

1-A

A folder handle from Windows — fires an event when any file appears inside it

Process Handle

1-B

An Access process handle from Windows — fires an event when Access closes

`WaitForMultipleObjects`

2

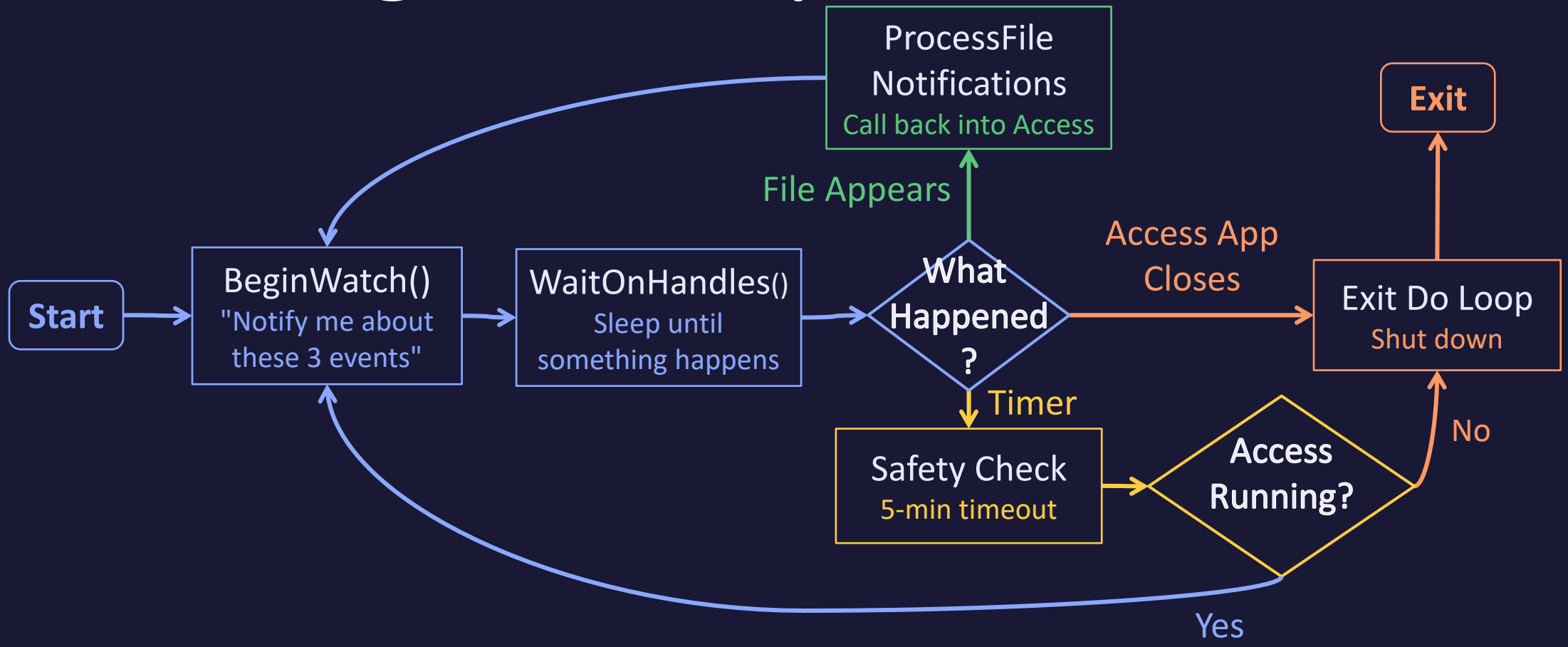
One API call that sleeps the thread until either handle fires (or a timer) — then acts

`GetObject` + `App.Run`

3

Finds the running Access instance and calls your VBA function

The Program Loop



How `WaitOnHandles` Works

Builds an array of two handles and passes them to one API call:

```
handles(0) = m_hEvent       ← directory change event
handles(1) = hProcess       ← Access process handle

result = WaitForMultipleObjects(2, handles(0), 0, timeoutMs)
```

Returns **which** handle woke it up:

`WAIT_OBJECT_0` (= 0) handle[0] fired → **a file appeared**

`WAIT_OBJECT_0 + 1` (= 1) handle[1] fired → **Access exited**

`WAIT_TIMEOUT` neither fired in 5 min → **safety check**

"Zero CPU while idle“: the thread is literally parked in the kernel scheduler.
It's the same mechanism Windows uses for its own services.

How the Callback Works

```
' Find the running Access instance via the Running Object Table
Set accessApp = GetObject(dbPath)

' Call the VBA function with the file path
accessApp.Run funcName, fileName

' Release immediately (milliseconds of contact)
Set accessApp = Nothing
```

`GetObject(dbPath)` is the key. When Access opens a database, it registers itself in Windows' **Running Object Table** (ROT) keyed by its file path. No process scanning. No window enumeration. Just a lookup.

The reference is grabbed, used, and released in milliseconds. Access barely knows it happened.

The Distribution Problem

You've built a utility `.exe` in twinBASIC. Now what?

Email it to users?

Email security blocks it.

Put it on a network share?

Two things to deploy instead of one.

Require an installer?

Overkill for a 200 KB file.

What if the `.exe` lived *inside* the `.accdb`?

Embedded Exe: usys_Resources

```
CREATE TABLE usys_Resources (
 ResourceName  TEXT(255) PRIMARY KEY,
 ResourceData  LONGBINARY
)

' One-time setup from Immediate Window:
ImportExe "C:\Build\FolderWatcher_win32.exe"
ImportExe "C:\Build\FolderWatcher_win64.exe"
ImportExe "C:\path\to\folderwatcher.ico"
```

The `usys_` prefix hides the table from the Navigation Pane.

Binary column holds raw bytes: `.exe`, `.ico`, `.dll`, anything (with no bloat).

The custom database icon uses the same pattern: extracted on startup, set as `AppIcon`.

Auto-Extraction at Runtime

```
User calls StartWatching("C:\Temp", "OnNewFileHandler")

Does FolderWatcher.exe exist next to the .accdb?
 Yes → launch it
 No  → read bytes from usys_Resources
       write to disk next to the .accdb
       launch it

Next time: file already there, extraction skipped.
```

One **.accdb** to distribute. Nothing to install. The user never knows the **.exe** exists.

Bitness-Aware Extraction

```
#If Win64 Then
  ExePath = CurrentProject.Path & "\FolderWatcher_win64.exe"
#Else
  ExePath = CurrentProject.Path & "\FolderWatcher_win32.exe"
#End If
```

Important nuance: Most twinBASIC utilities won't need this. A standalone `.exe` runs in its own process, so a single 32-bit build works fine with both 32-bit and 64-bit Access.

FolderWatcher is the exception because it calls `GetObject` to get a COM reference back into Access. COM interop requires the caller and the host to match bitness.

Out-of-process = bitness doesn't matter. COM callbacks = it does.

GBLauncher: Why bother?

One Workflow, Every Client

Same process for full VPN, limited remote, or no network access.

No Infrastructure Required

Dropbox share links only — no file servers, no VPN, no agents.

Auto-Trusted Locations

Registers cache folder before launch — no "Do you trust this?" prompts.

Smart Caching & Offline Fallback

Instant launch when unchanged; last cached version works offline.

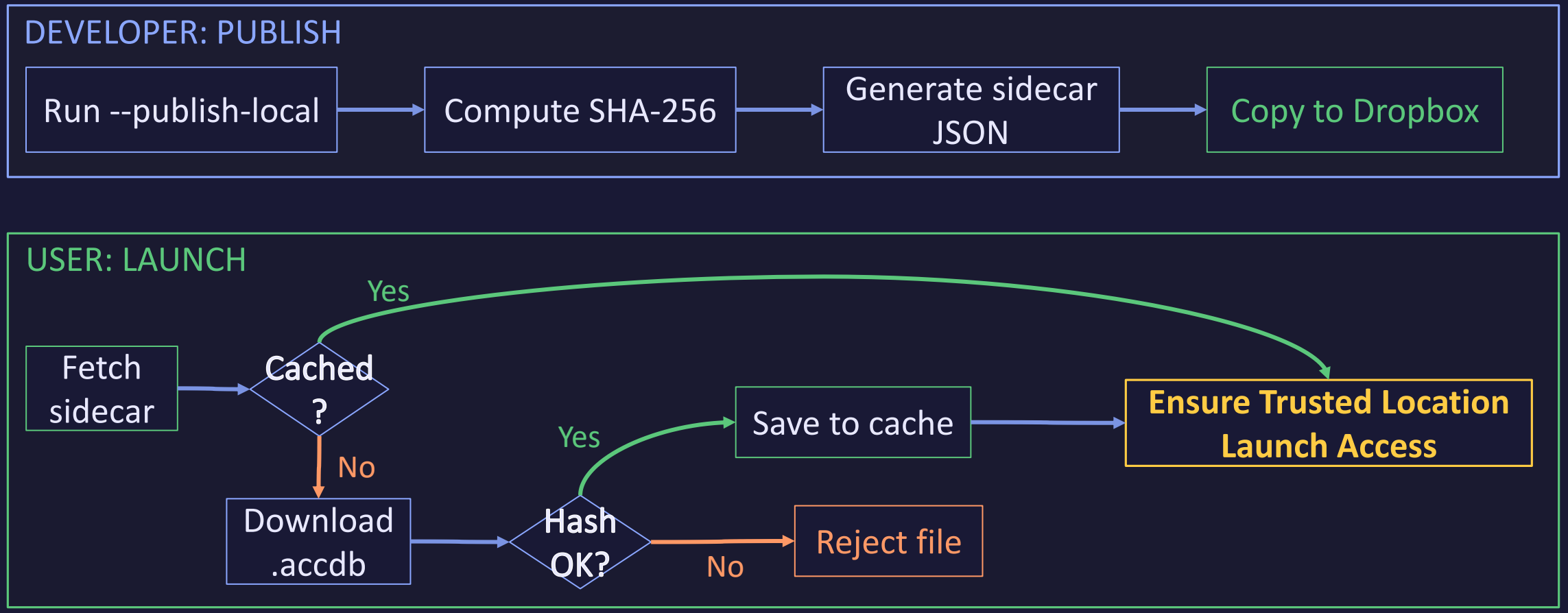
No Admin Privileges Required

%LOCALAPPDATA% and HKCU only — install and update without IT.

Professional Desktop Shortcut

Branded icon that checks, downloads, and launches in one click.

GBLauncher: Overview



GBLauncher: Why in twinBASIC?

Zero Dependencies

Single .exe under 500 KB — runs on any Windows 7–11 machine

Direct Win32 API

WinHTTP, CNG, Registry, COM — direct calls, no wrappers

First-Class COM

CreateObject is one line — no other modern language makes it this easy

Familiar VBA Syntax

Read, modify, and extend the source on day one

One Codebase, Many Builds

Conditional compilation — client, dev x86, dev x64 from one source

Trivial Self-Update

Download, verify hash, replace, relaunch — it's just one file

GBLauncher Recap

The developer runs `--publish-local` to create a release.

The user double-clicks a shortcut.

GBLauncher handles everything in between:

Trusted locations.
Verification.
Download.
Caching.
Launch.

twinBASIC Licensing (Subscription)

Community	Professional	Ultimate
FREE	**GBP £26|EUR €29|USD $35 /mo***	**GBP £39|EUR €45|USD $52 /mo***
Create 32/64-bit EXE/DLLs	All Standard features plus:	All Professional features plus:
Splash screen for 64-bit	No splash screen on 64-bit EXE/DLL	Private product support via email
Windows only (unoptimized compilation)	Optimized compilation (still Windows-only)	Cross-platform support (Linux/Mac/Android)

* Pricing shown reflects a 35% pre-order discount (discounted forever)

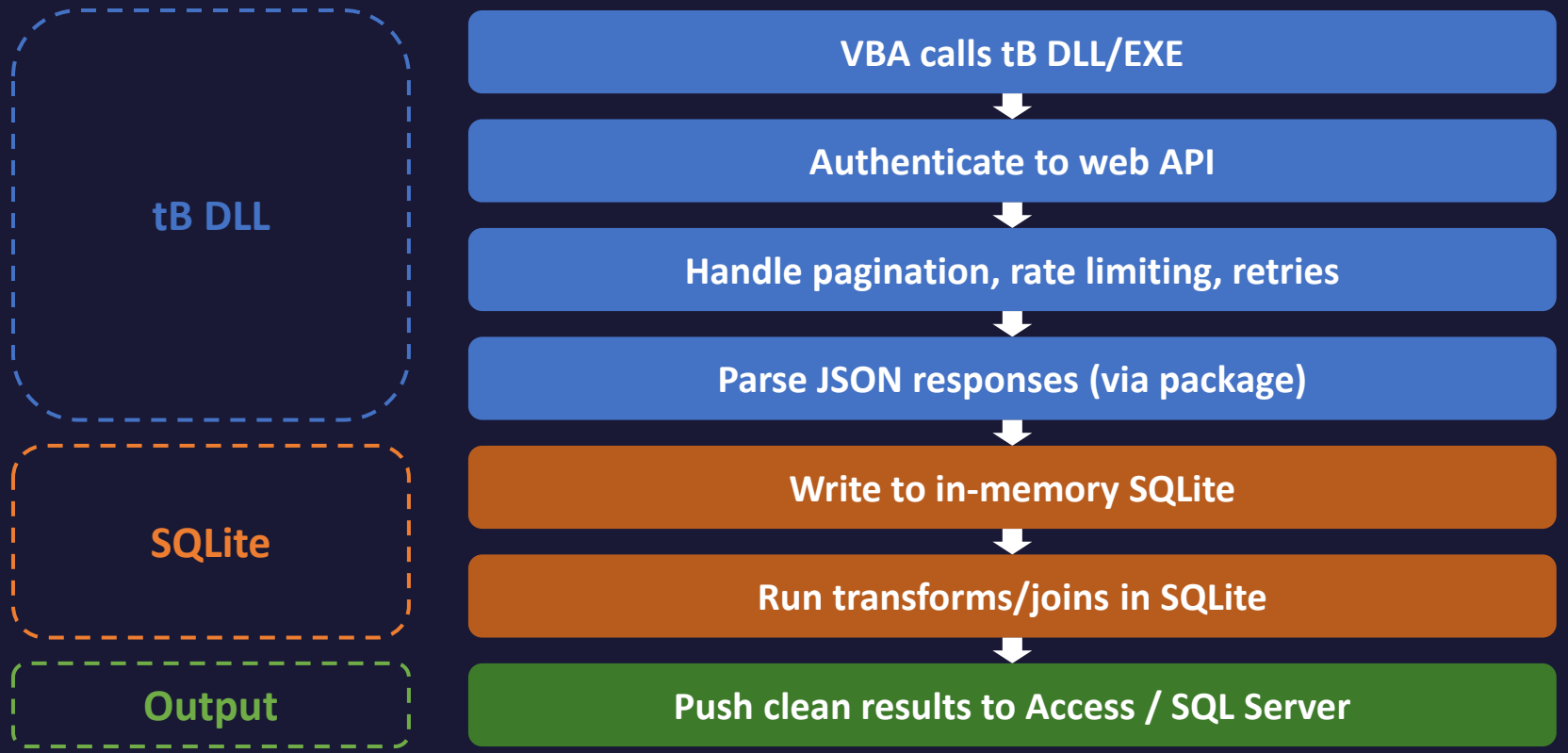
- Commercial use, royalty-free license for *all editions*
- VIP Gold Lifetime License: **GBP £5000**

Questions?

Links and Resources:

nolongerset.com/devcon-2026/

twinBASIC Web Data API Architecture



Bonus: Karma Kargo

Built from scratch in 28 days

twinBASIC + AI (Claude)

- 55 architecture documents → working game
- <2% of source code read by a human

Custom GDI+ rendering • 3 SQLite databases
Generic ORM • Seeded randomness (a la Minecraft)
Single .exe • Zero dependencies

karmakargo.com