

Great Migrations

Billing & Audit Synchronization System (VB6 to C#)

Billing & Audit Synchronization System (BASS)

Client

Client is a major provider of systems that help enterprises manage the full lifecycle of their utility data, from bill payment to energy procurement and sustainability reporting.

Legacy Scope

BASS is an internal system that synchronizes billing and audit data across the databases behind this work, a set of unattended “auditor” processes that move, transfer, and reconcile records around the clock.

*BASS ran on **Visual Basic 6, COM/COM+, and ActiveX** — a platform Microsoft retired from mainstream support long ago, and one that is increasingly difficult to staff, secure, and run on modern Windows. Great Migrations was engaged to reproduce BASS’s functionality on **C# / .NET Framework 4.8.1** as a system that could be, in the words of the statement of work, “productively maintained on .NET for the long term.”*

*The application comprised a family of four unattended “auditor services” that, between them, **import, transfer, reconcile, and synchronize** billing and audit records across the client’s systems on a continuous, scheduled basis with no operator interaction.*

*The BASS codebase was comprised of **18 inter-related VB6 projects** (13 ActiveX DLLs, 4 out-of-process ActiveX EXEs, and 1 EXE) referencing **88 VB6 source files**, 39 classes, 19 forms, and 12 modules, containing approximately **15,000 lines of code** and using **12 COM libraries and components (4,337 references, ADODB alone accounting for 3,362)** plus **129 Win32 API calls**. Configuration was split across the Windows Registry and INI files; logging*

wrote to a database table with pop-up dialogs; credentials were protected by a home-grown password cipher; and the system carried **no automated tests**. A key consideration of the project was that the legacy system had been working reliably for many years, so experience and knowledge of the system was very limited.

Modernization Approach

Great Migrations applied its **Tool-Assisted Rewrite methodology**: the in-scope codebase was first translated to C# with gmStudio, faithfully and without translation errors, to establish a reviewable baseline. From there the team progressed through **four delivery milestones**, moving beyond a literal translation. A more contemporary system architecture was designed and developed by hand, then deterministic transformation rules rewrote the legacy code and integrated it into the new framework. Corrections and patterns were applied at the **translation-rule layer** rather than by hand-editing generated code, so they persist across re-translation and carry forward as reusable assets to future migrations. The translation orchestration was implemented using gmStudio's new parallel processing capability allowing the team to complete the full system upgrade: all VB6-to-C# translations and .NET builds in less than 20 seconds*.*

Technical Highlights

The most significant work restructured how the software runs, not just what language it is written in. The UI-driven, COM-activated desktop host was re-architected into **four independent, headless Windows Services**, each a **~30-line composition root** on a shared, reusable service framework:

Highlight	Details
Execution model rebuilt	WinForms message-pump timers and late-bound CreateObject COM plugins were replaced by a .NET Generic Host, a BackgroundService interval loop, dependency-injected typed IProcess interfaces (runtime failures become compile-time safety), and cooperative shutdown under the Windows Service Control Manager.

Highlight	Details
Obsolete UI 19 GUI forms and 224 controls	<i>The obsolete UI code was removed. Application is now managed as Windows services.</i>
Configuration consolidated:	<i>distributed Registry + INI access reworked to use more standard access to a single appsettings.json per service on Microsoft.Extensions.Configuration, with startup validation that fails a misconfigured service visibly rather than letting it run broken. (Modern configuration management was added above the original contract scope because it materially improves long-term supportability.)</i>
Structured logging consolidated:	<i>A SQL log table and various pop-up dialogs were modified to use Serilog rolling files with 30-day retention, behind a framework-neutral logging abstraction.</i>
Security hardened:	<i>Registry-assembled connection strings and a stored, custom-encrypted password to a single config-sourced connection string using Integrated Windows Authentication with no stored credentials.</i>
Legacy dependencies retired:	<i>The entire 4,337-reference COM footprint was reduced to lightweight .NET replacements; the production deploy tier carries no COM+, MAPI, Outlook, DAO, or Scripting dependency, and Win32 usage in the application code dropped from 129 P/Invoke calls to none.</i>

*The delivered application logic is now roughly **12,500 lines of idiomatic, verified C#**, and independent Visual Studio metrics rate the new code “**maintainable**” (Maintainability Index 74.6, assembly average).*

Engineering & DevOps Highlights

A system is only maintainable long-term if it rests upon reliable machinery to build, test, and ship it. Great Migrations and the client analyzed the legacy built new support in line with their new architecture:

Area	Details
Automated test harness	a Reqnroll + xUnit BDD suite that exercises the real auditor plugins in-process against a live SQL database, including a source-controlled fault-injection system for bad/missing configuration that is compiled out of customer builds.
Developer test tooling	a scripted, repeatable Pester (PowerShell) test-automation fleet covering every functional area, with cumulative and ad-hoc run modes and a database revert/restore ledger, giving developers a one-command path to a green, idempotent test run on any machine.
Code-coverage reporting	Coverlet + ReportGenerator (Cobertura → HTML), scoped to the application assemblies, establishing an honest first coverage baseline where there was no measurement at all.
Installer	WiX 7 authoring producing a true 64-bit MSI that registers each service with the Service Control Manager and sets auto-start and auto-restart-on-failure policies, replacing manual, per-machine COM registration.
CI/CD pipeline:	an Azure DevOps pipeline that restores, builds the solution Release/AnyCPU, versions and packages the MSI installer, and publishes the artifacts.

Verification & Refinement Highlights

Correctness was proven against a signed-off test plan whose pass criterion was that .NET database output match the VB6 baseline exactly, with a performance requirement to match or beat the legacy system. The end to end process was automated for .NET including data set up, test case execution, results inspection, and reporting over 400 test points.

Project Summary

	<i>Before — VB6 BASS</i>	<i>After — .NET BASS Services</i>
<i>Execution model</i>	<i>1 desktop host + timers, COM plugins</i>	<i>4 headless Windows Services, DI, SCM- managed</i>
<i>User interface</i>	<i>19 forms, 224 controls</i>	<i>None</i>
<i>COM / Win32</i>	<i>4,337 COM refs · 129 P/Invokes</i>	<i>.NET Replacements · no COM Interop and no Win32</i>
<i>Configuration</i>	<i>Registry + INI</i>	<i>appsettings.json through Microsoft.Extensions.Configuration</i>
<i>Logging</i>	<i>DB table + pop-ups</i>	<i>Structured Serilog, 30-day retention</i>
<i>Security</i>	<i>SQL Authentication</i>	<i>Integrated Windows Authentication</i>
<i>Tests / coverage / install / CI</i>	<i>None</i>	<i>BDD suite - coverage baseline - WiX MSI - Azure pipeline - Pester Batch Test Automation</i>

*Delivered over a ~22-week program (March-July 2026), the four production services are packaged in a MSI installer and the system is **ready for User Acceptance Testing and production cut-over.***