TECHNICAL RELEASE NOTES

VB6 AI Migrator

Comprehensive release notes & capabilities — versions 6.5 to 9.5+

The complete technical history and capabilities guide for the VB6 AI Migrator (formerly the Visual Basic Upgrade Companion). It details every architectural enhancement, code-generation improvement and supported upgrade option through modern .NET 8 — written for enterprise architects and engineering leaders evaluating the deterministic modernization of legacy Visual Basic 6.0 applications.

CONTENTS BY VERSION

Version 6.5

Foundational compilation, performance & fingerprinting

Application fingerprint

The license permits modernization of a specific application. To let teams refactor and optimize legacy code before migrating, v6.5 introduced application fingerprinting: you can safely modify the app and rerun the migration, and the engine verifies the modified codebase's fingerprint matches the original license footprint.

Enhanced compilation & migration performance

Across several million lines of real-world code, post-migration compile errors were reduced by an average of **84%**, and net migration processing time improved by **2–5×** on large-scale enterprise applications.

References resolution & shared files

- **Automated resolution:** complex project and class references that once required manual intervention are now located and resolved automatically.
- **Shared files:** projects using shared files migrate with clean, idiomatic code — removing clumsy namespace prefixes such as `X = MySolution_Shares.Module1.MyEnum.field1`.

Data-access helper classes

The `FieldHelper`, `FieldsHelper` and `RecordsetHelper` classes were refactored for strict OO compliance (abstraction and encapsulation). Fields can be accessed via keys not distinctly identifiable as integers or strings across ADODB, ADOR and DAO. Supported members include `Recordset.Fields`, `TableDef.CreateField`, `Fields.Append/Delete/Count/Item`.

Managed type libraries & collection keys

- **Managed TLBs:** better recognition of managed TLB libraries created in .NET but consumed by VB6; COM interop expressions such as `get_MyProp()` translate natively to .NET properties (`MyProp`).
- **Collection keys:** VB6 generic collections accessed via unpredictable parameter types (like `Variant`) are handled automatically.
- **Menu-item control arrays:** arrays mixing menu items and separators generate distinct `ToolStripItem` and `ToolStripSeparator` arrays.

Version 7.0

VB6 independence & ByteInsight integration

VB6 dependency removal

The Migrator no longer requires an active Visual Basic 6.0 installation on the host. It natively ships with metadata from popular VB6 libraries, severing the operational risk of maintaining obsolete environments.

Custom & third-party libraries (TLBs)

For proprietary or missing binary dependencies, users can extract metadata into Type Libraries (TLBs) and import them directly — ensuring accurate code generation even when original binaries are unavailable.

ByteInsight assessment enhancements

- Windows APIs are specifically counted and reported.
- Designer-created controls are identified.
- Unsupported legacy files (WebClasses, Crystal Reports) are flagged.
- Potential internal references are marked for downstream resolution.

Migration improvements

- Method return types can be declared as multi-dimensional arrays.
- The deprecated `FixedLengthString` arrays class was eradicated; struct-based fixed-length strings now resolve via native .NET patterns.
- File-overwrite bugs for identical names with different extensions were fixed.

Version 7.1

Parallel processing & code readability

Parallel migration engine

The upgrade step can now run in parallel: the engine divides multi-project solutions across available processor cores, reducing upgrade time to **40–60%** of sequential execution.

Readability & structural enhancements

- Improved recognition and removal of unnecessary `ByRef` parameters.
- Compilation errors from parameterless events eliminated.
- Shared-files migration handles varying parameter resolutions across projects natively.
- Array dimensions, declarations and initializations align strictly with C# and VB.NET best practices.
- Complex string concatenations mixing operand types and `+` operators are safely coerced for compilation equivalence; the `Cursor` enum translates perfectly to native .NET classes.

Enhanced library conversions

Upgraded mapping for `SSDataWidgets`, `TrueDBGrid`, `DataSource` members, `CommonDialog` patterns and specific `OLE_Color` properties. Optional `SSPanel` → `Label/PictureBox` conversion was added.

Version 7.2

Visual Studio 2017, scalability & advanced typing

Visual Studio 2017 & typing inference

Generates fully compliant Visual Studio 2017 source and project/solution files. The AI-based typing-inference engine was optimized for memory efficiency, allowing exceptionally large monolithic applications to be fully analyzed and migrated.

Advanced language features

- **Multidimensional arrays:** mutating VB6 array dimensions map into the strictest, most representative .NET structures.
- **COM Let/Set properties:** improved typing analysis for parameters and return types from COM `Let` and `Set` properties.
- **L-value vs. R-value:** heuristics updated so left-side expressions inherit types from right-side expressions during assignment.

Increased mapping coverage

Added mappings for Accusoft classes, Crystal Reports (`CRAXDRT_CRVIEWERLibCtl`), National Instruments (`CWUIControlsLib`), FarPoint memos (`MemoLibfpMemo`) and `OracleInProc` database connections to .NET data-connection classes.

Version 8.0

Enterprise memory models & ByRef/ByVal intelligence

Performance at scale

A new memory model processes monolithic applications (`> 2,000,000` lines). Parallel conversion handles up to **12 projects simultaneously**, tested across 24M+ lines of real-world code.

Advanced code generation

- **Shared Files Bridge:** a shared file could act as a method in one module and a property in another; the platform now generates a bridge class so each project gets its correct implementation.
- **Drag and drop:** controls relying on drag-and-drop migrate to native .NET event handlers.
- **Intelligent ByRef → ByVal:** VB6 defaults to `ByRef`; v8 identifies when arguments can be safely passed `ByVal`, producing highly readable C#.
- **Event renaming:** event handlers are renamed automatically and consistently to prevent cascading compile errors.

Version 8.1

Deep syntax refactoring & complex initializations

Collection type inference for inconsistent usages

Dynamically mixed types added to generic collections translate into .NET `OrderedDictionary` implementations.

```
C#

OrderedDictionary x = new OrderedDictionary(System.StringComparer.OrdinalIgnoreCase);

x.Add("Value1", "1");

x.Add("Value2", 2);

string nodeKey = "Value2";

string myTxt = nodeKey.Substring(0, Math.Min(

    ReflectionHelper.GetPrimitiveValue<int>(x[nodeKey]), nodeKey.Length));

MessageBox.Show(myTxt, Application.ProductName);
```

Intrinsic alignment support

VB6 alignments translate cleanly to .NET `HorizontalAlignment` enumerations within `switch` statements.

```
C#

switch (UpgradeHelpers.Gui.TextAlignmentHelper.ToHorizontal(Label7.TextAlign))

{

    case HorizontalAlignment.Center:

        MessageBox.Show("Alignment is Center", Application.ProductName);

        break;

    case HorizontalAlignment.Left:

        MessageBox.Show("Alignment is Left Justify", Application.ProductName);

        break;

    // ...

}
```

Array declaration & parameterless constructors

```
C#

// Array Initialization

string[] fixedstring = ArraysHelper.InitializeArray<string[]>(new int[]{10}, new object[]{5});



// Constructor Initialization

ADORecordSetHelper rs = new ADORecordSetHelper("");
```

Optional parameters & decimal casting

Coerces VB6 `IsMissing()` boolean logic and strictly casts legacy numerical checks into .NET `Decimal (m)` types.

```
C#

object strLOC1 = null;

if ((ReflectionHelper.GetPrimitiveValue<decimal>(strLOC1) >= 11 &&

     ReflectionHelper.GetPrimitiveValue<decimal>(strLOC1) <= 11.9m))

{

    // Execution block

}
```

Singular-name enumerations & nested loops

Resolves syntax errors when VB6 enums share singular namespace collisions. Deeply nested loops containing `Exit For` or `Exit Do` correctly generate target `goto` break labels so the exact intended loop is exited.

Versions 8.2 & 8.3

.NET Core 3.1 support, NuGets & local functions

Support for .NET Core 3.1

Version 8.3 delivered the ability to target **.NET Core 3.1** (C#, VS 2019), letting legacy Windows applications move toward cross-platform, cloud-native architectures.

Upgrade-option filtering for .NET Core

Because .NET Core deprecated several legacy Microsoft assemblies, the Migrator intelligently disables incompatible upgrade options. Supported .NET Core mappings include:

Category	Upgrade option / feature	.NET Core
Data Access	ADODB to ADO.NET using System.Data.Common	**Yes**
Data Access	ADODB/DAO/RDO/OracleInProc to COM Interop	**Yes**
Data Access	SQLDMO to Microsoft.SqlServer.Smo	**Yes**
Grids	MSDataGridLib to Windows.Forms.DataGridView	**Yes**
Grids	MSFlexGrid to DataGridViewFlex helper class	**Yes**
Grids	TrueDBGrid to ComponentOne TrueDBGrid (.NET)	**Yes**
Microsoft	MSComCtl2 / MSComctl to native System.Windows.Forms	**Yes**
Microsoft	MSMask to System.Windows.Forms.MaskedTextBox	**Yes**
Microsoft	MSXML2 to System.Xml classes	**Yes**
Microsoft	SHDocVw to System.Windows.Forms.WebBrowser	**Yes**
Microsoft	Shell32 to System.Diagnostics methods	**Yes**
Code Conv.	Generate auto-implemented properties (C#)	**Yes**
Code Conv.	Convert error handling to try-catch (with lambdas)	**Yes**
Sheridan	SSActiveTabPanel to System.Windows.Forms.TabControl	**Yes**
Sheridan	SSActiveToolBars to System.Windows.Forms.ToolStrip	**Yes**
Sheridan	SSSplitter to System.Windows.Forms.SplitContainer	**Yes**

GoSub to local functions

`GoSub...Return` statements branch to and return from subroutines within a procedure; the platform now supports this elegantly via C# local functions.

```
C#

public void Foo(int p)

{

    string value = "";

    if (p == 1) { Lbl1(); } else { Lbl2(); }

    MessageBox.Show(value);

    return;



    void Lbl1() { value = "First value"; }

    void Lbl2() { value = "Second value"; }

}
```

Helpers as NuGets & logging

For migrations targeting .NET Framework 4.7.2+ (including .NET Core), helper classes integrate natively as NuGet packages rather than binary DLLs. Logging was expanded to include Debug, Info, Warning and Error tracing.

Versions 9.0 – 9.4

.NET 5 & 6, multiple GUID resolution & SDK projects

.NET 5 & .NET 6 support

Targets expand to **.NET 5 and .NET 6 (LTS)** for C# and VB.NET. Project files use the modern SDK-style `.csproj` format, omitting temporary migration logs and package folders from the solution hierarchy.

Multiple GUID reference resolution (v9.4)

Previously, projects using disparate versions of a reference (e.g. `MSXML2` v3 vs v6) forced a global upgrade to the highest version. v9.4 lets disparate versions coexist, generating the exact COM interop each project module requires.

RDO to .NET Core / .NET 5

Remote Data Objects (RDO) references, previously limited to COM interop when migrating to Core, can now migrate fully to native .NET Core 3.1 or .NET 5 data classes.

Font behavior inheritance

Replicates VB6's exact UI rendering: if a .NET control lacks a specific `Font` property during migration, it correctly inherits the font from its VB6-designated container (Form or User Control).

Version 9.5 & beyond

.NET 7, .NET 8 & modern C# language features

.NET 7 & .NET 8 support

The Migrator supports **.NET 7 and .NET 8** platforms, available when selecting Visual Studio 2022 as the target.

Missing-references feature

If a target environment lacks legacy VB6 libraries, the platform automatically resolves missing references by downloading and generating a `.tlb` metadata file — so code generation proceeds flawlessly even on modern 64-bit operating systems.

GoTo support for C#

Complex `GoTo` patterns and error-handling routines that previously threw warnings are now structurally translated into resilient try-catch blocks and local functions, preserving exact execution paths.

Advanced C# feature implementation

To keep generated code idiomatic, the engine incorporates modern C# (8–10) syntax:

Discards — unused variables use discard syntax (`_`) to signal clear intent.

```
C#

int a = 5, b = 6, c = 10;

_ = a + b + c; // Discarded

_ = UsingDiscardParameteres(10, "Test", false);
```

Interpolated strings (C# 10) — concatenations and constants targeting .NET 6+ upgrade to readable interpolated strings.

```
C#

const double bookPrice = 24.99d;

const string BookName = "The Lord of The Rings";

string str02 = $"Book '{BookName}' price is {bookPrice.ToString()}";

string fullName = $"{Name} {lastName}";
```

Relational pattern matching (C# 9+) — legacy `Select Case` statements use relational patterns for elegant, high-performance logic.

```
C#

internal static int GetTax2(Product p)

{

    switch (p.CategoryID)

    {

        case 0 or 1:        return 0;

        case >= 2 and <= 5: return 5;

        case > 20:          return 15;

        default:            return 10;

    }

}
```

Conclusion

The most advanced, actively maintained deterministic engine

From early structural refactoring to generating C# 10 features for .NET 8 cloud-native architectures, the **VB6 AI Migrator** is the industry's most advanced, actively maintained deterministic transformation engine — ensuring your legacy transition is fast, predictable and technically flawless.

gapvelocity.ai