• Professional Coding Tools • Write More Solid Code • Reduce Common Coding Drudgery • Add Sophisticated Error Handling • Understand & Fix Bugs Faster •

Total Visual CodeTools™

Supports Access 2016 through 2000

FREE TRIAL

Simplify Code Writing, Clean Up, and Delivery

Why Total Visual CodeTools Exists

Total Visual CodeTools was created to address the challenges we experienced in our own efforts to write great VB6 and VBA (Office/Access/Excel) applications. From writing new code, to taking over someone else's work (or work we wrote years ago that looks like someone else wrote it), to delivering solutions that are easier to support, VBA/VB6 offers the capability to do so, but not automatically.

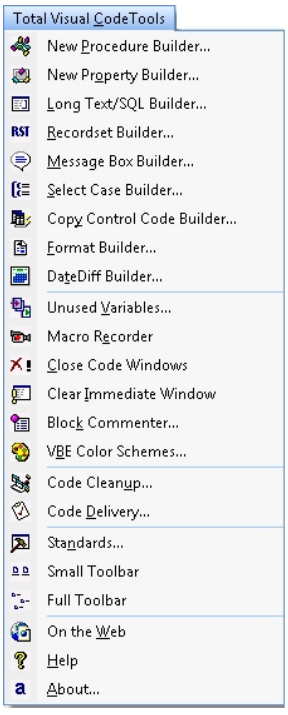
Leveraging our technology to parse VBA/VB6 code, Total Visual CodeTools not only creates new code but updates existing code in powerful ways. It lets you apply Best Practices to your projects and standardize existing solutions.

The result is your ability to deliver more professional solutions, quicker than ever, and in ways that would be nearly impossible to do manually. Discover why so many developers insist on using it.

New Features!

- Supports Office/Access 2016, plus 2013 thru 2000
- Immediate Overwrite feature for Cleanup and Delivery
- Copy Control Builder can set multiple target controls
- Long Text Builder retrieves SQL plus space/tab removal
- Select Case Builder supports text blocks and data ranges
- Enhanced Recordset Builder
- Resizable forms
- Improved standards storage

Deliver More Professional Solutions Faster!



Code Builders Simplify Code Writing

The Code Builders let you quickly create new procedures, open recordsets using ADO or DAO, convert SQL strings to variables while handling quotes and smart word-wrapping, visually create message boxes, and much more.

Code Cleanup Standardizes Code

Cleanup existing code: add Option Explicit to modules that lack it, add error handling to procedures without it, rename variables to your naming conventions, standardize line indentations of loops, sort procedures, etc.

Code Delivery Deploys Better Solutions

Add line numbers so your error handler can pinpoint exactly where a crash occurs. Simplify support by reducing the need to get information or repro steps from users!

"Total Visual CodeTools is by far my favorite third-party product"
Alison Balter, Trainer, Author of Mastering Access Development

Create Quality Code Consistently

Have your entire development team share and enforce a consistent set of coding standards. Establish quality directly into your processes!

FMS • fms@fmsinc.com • fmsinc.com • 703-356-4700

FMS

Microsoft Partner

1998 Access Office VB Advisor Readers' Choice Silver — Best Access Add-In

2000 Tips & Tricks Reader Choice Award — Reader's Choice

2001 Access VB SQL Editors' Choice Advisor.com Gold — Best VB Add-In

Supports
VB 6.0, Access, Office 2016, 2013/2010 (32-bit), 2007, 2003, 2002, 2000

Single $299 5-Pack $899
Upgrade Single $199

30 DAY GUARANTEE MONEY BACK

• Copy Event Procedures Between Controls • Easily Create Class Properties • Simplify SELECT CASE and the DateDiff and Format functions • Close All Windows •

• Add Procedure Lists to Modules • Replace Error Handing • Define Your Own Variable Naming Convention Based on Global or Local Scope • Apply Naming Conventions • Compact and Obfuscate Code • Find Unused Variables •

• New Procedure Builder Inserts Your Commenting and Error Handling Structures • Long Text/SQL Builder Assigns SQL to Code • Recordset Builder Creates ADO and DAO Connections • Visually Create Message Boxes •

Total Visual CodeTools for Office/VBA and Visual Basic 6.0

Code Cleanup

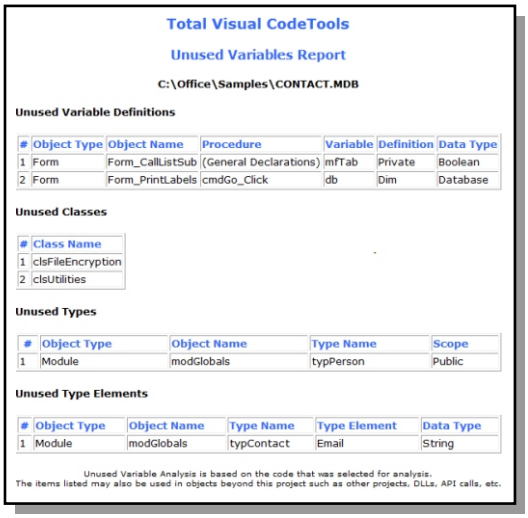
Total Visual CodeTools transforms poorly formatted and hard-to-read code to your standards for commenting, error handling, indentations, blank lines, variable naming conventions based on data type and scope, procedure sort order, etc.

Original:

```
Function AddRows(CustId As Long, retval As Integer) As Boo

Dim Northwind As DAO.Database
Dim Customers As DAO.Recordset
Dim LastName1, LastName2, LastName3 As String

LastName1 = "Jones"
 LastName2 = "Smith"
  LastName3 = "Gates"

Set Northwind = Workspaces(0).OpenDatabase("c:\nwind.mdb")
  Set Customers = Northwind.OpenRecordset("customers")
Customers.FindFirst ("CustomerID=" & CustId)

If Customers.Fields(0).Value = LastName1 Then AddRows = Tr


Select Case Customers.Fields(1).Value
Case 1: retval = 12
  Case 2: retval = 13
    Case 3: retval = 14
   End Select
End Function
```

Cleaned Up:

```
Add naming conventions to the procedure declaration with parameter prefix:
Function AddRows(plngCustId As Long, pintRetval As Integer) As Boolean
  Add procedure comments with all parameters, the return value and dates:
  ' Comments:
  ' Params  : plngCustId
  '           pintRetval
  ' Returns : Boolean
  ' Modified: 06/01 LC

  Add Error Handling Enabler:
  On Error GoTo PROC_ERR

  Apply naming conventions to variable names based on data type:
  Dim dbsNorthwind As DAO.Database
  Dim rstCustomers As DAO.Recordset
  Split multiple Dims in the same line to separate lines:
  Dim varLastName1
  Dim varLastName2
  Dim strLastName3 As String

  Standardize indentation and line spacing:
  varLastName1 = "Jones"
  varLastName2 = "Smith"
  strLastName3 = "Gates"

  Set dbsNorthwind = Workspaces(0).OpenDatabase("c:\nwind.mb")
  Set rstCustomers = dbsNorthwind.OpenRecordset("customers")
  rstCustomers.FindFirst ("CustomerID=" & plngCustId)
  Fix single-line If statements to add End If:
  If rstCustomers.Fields(0).Value = varLastName1 Then
    AddRows = True
  End If
  Eliminate extra blank lines:

  Fix colon-separated statements and standardize indentation:
  Select Case rstCustomers.Fields(1).Value
    Case 1
      pintRetval = 12
```



Total Visual CodeTools

Unused Variables Report

C:\Office\Samples\CONTACT.MDB

Unused Variable Definitions

#	Object Type	Object Name	Procedure	Variable	Definition	Data Type
1	Form	Form_CallListSub	(General Declarations)	mfTab	Private	Boolean
2	Form	Form_PrintLabels	cmdGo_Click	db	Dim	Database

Unused Classes

#	Class Name
1	clsFileEncryption
2	clsUtilities

Unused Types

#	Object Type	Object Name	Type Name	Scope
1	Module	modGlobals	typPerson	Public

Unused Type Elements

#	Object Type	Object Name	Type Name	Type Element	Data Type
1	Module	modGlobals	typContact	Email	String

Unused Variable Analysis is based on the code that was selected for analysis.
The items listed may also be used in objects beyond this project such as other projects, DLLs, API calls, etc.

Unused Variable Analysis

Find code that's defined but never referenced or used:

- Unused variables defined at the procedure, module, and project/global levels through Dim, Public, and Private
- Unused procedure parameters
- Unused constants
- Unused classes
- Unused user defined types and their elements

Unused code often occurs when eliminating code without noticing the variable definition remained. Fix this by simply deleting the variable definition.

This can also occur if you define a variable then forget to use it. Maybe you wanted to track something or do something special. Rather than just deleting the variable, determine its purpose and implement the feature or delete it if it's no longer necessary.

New Procedure Builder

The New Procedure Builder lets you quickly create procedures with the styles and structure you want. Dynamically design your procedure, watch it being written, then insert it in your module. Easily specify:

- Scope: Public or Private?
- Type: Sub or Function with return type assignment
- Custom comment structures referencing the current date, time, and your initials
- Error handling routines customized for modules, classes, and forms/reports while referencing the current module and procedure names, etc.
- Option to only use the body for insertion into existing event procedures