Development and Maintenance Tools for
Microsoft® Visual Basic, Access, Office, and VBA Developers

Total Visual CodeTools™

For Microsoft Office
and Visual Basic 6.0

www.fmsinc.com

License Agreement

PLEASE READ THE FMS SOFTWARE LICENSE AGREEMENT. YOU MUST AGREE TO BE BOUND BY THE TERMS OF THIS AGREEMENT BEFORE YOU CAN INSTALL OR USE THE SOFTWARE.

IF YOU DO NOT ACCEPT THE TERMS OF THE LICENSE AGREEMENT FOR THIS OR ANY FMS SOFTWARE PRODUCT, YOU MAY NOT INSTALL OR USE THE SOFTWARE. YOU SHOULD PROMPTLY RETURN ANY FMS SOFTWARE PRODUCT FOR WHICH YOU ARE UNWILLING OR UNABLE TO AGREE TO THE TERMS OF THE FMS SOFTWARE LICENSE AGREEMENT FOR A REFUND OF THE PURCHASE PRICE.

Ownership of the Software

The enclosed software program ("SOFTWARE") and the accompanying written materials are owned by FMS, Inc. or its suppliers and are protected by United States copyright laws, by laws of other nations, and by international treaties. You must treat the SOFTWARE like any other copyrighted material except that you may make one copy of the SOFTWARE solely for backup or archival purpose, and you may transfer the SOFTWARE to a permanent storage device.

Grant of License

The SOFTWARE is available on a per license basis. Licenses are granted on a PER USER basis. For each license, one designated person can use the SOFTWARE on one computer at a time.

Other Limitations

Under no circumstances may you attempt to reverse engineer this product. The SOFTWARE is licensed as a single product and may not be separated by use for more than one user at a time. You may not rent or lease the SOFTWARE.

You may not transfer any of your rights under the FMS Software License Agreement to other individuals or entities. Without prejudice to any other

rights, FMS may terminate this FMS Software License Agreement at any time if you fail to comply with any of its terms. In such an event of termination, you must destroy and stop using all affected SOFTWARE copies.

Transfer of License

If your SOFTWARE is marked “NOT FOR RESALE,” you may not sell or resell the SOFTWARE, nor may you transfer the FMS Software license.

If your SOFTWARE is not marked “NOT FOR RESALE,” you may transfer your license of the SOFTWARE to another user or entity provided that:

1. The recipient agrees to all terms of the FMS Software License Agreement.
2. You provide all original materials including software disks or compact disks, and any other part of the SOFTWARE’s physical distribution to the recipient.
3. You remove all installations of the SOFTWARE.
4. You notify FMS, in writing, of the ownership transfer.

Limited Warranty

If you discover physical defects in the media on which this SOFTWARE is distributed, or in the related manual, FMS, Inc. will replace the media or manual at no charge to you, provided you return the item(s) within 60 days after purchase.

ALL IMPLIED WARRANTIES ON THE MEDIA AND MANUAL, INCLUDING IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE LIMITED TO SIXTY (60) DAYS FROM THE DATE OF PURCHASE OF THIS PRODUCT.

Although FMS, Inc. has tested this program and reviewed the documentation, FMS, Inc. makes no warranty or representation, either expressed or implied, with respect to this software, its quality, performance, merchantability, or fitness for a particular purpose. As a result, this software is licensed “AS-IS,” and you are assuming the entire risk as to its quality and performance. IN NO EVENT WILL FMS, INC. BE LIABLE FOR DIRECT, INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES RESULTING FROM THE USE, OR INABILITY TO USE THIS SOFTWARE OR ITS DOCUMENTATION.

THE WARRANTY AND REMEDIES SET FORTH IN THIS LIMITED WARRANTY ARE EXCLUSIVE AND IN LIEU OF ALL OTHERS, ORAL OR WRITTEN, EXPRESSED OR IMPLIED.

Some states do not allow the exclusion or limitation of implied warranties or liability for incidental or consequential damages, so the above limitations or exclusions may not apply to you. This warranty gives you specific legal rights; you may also have other rights that vary from state to state.

U.S. Government Restricted Rights

The SOFTWARE and documentation are provided with RESTRICTED RIGHTS. Use, duplication, or disclosure by the Government is subject to restrictions as set forth in subparagraph (c) (1) (ii) of the Rights in Technical Data and Computer Software clause at DFARS 252.227-7013 or subparagraphs (c) (1) and (2) of the Commercial Computer Software - Restricted Rights at 48 CFR 52.227-19, as applicable.

Manufacturer is FMS Inc., Vienna, Virginia.
Printed in the USA.

Acknowledgments

We would like to thank everyone who contributed to make Total Visual CodeTools a reality. Thanks to the many existing users who provided valuable feedback and suggestions, and to all of our beta testers for their diligence and feedback.

Many people at FMS contributed to the creation of Total Visual CodeTools, including:

- **Product Design and Development:** Luke Chung
- **Production, Marketing, and Graphics:** Mellenie Runion and Luke Chung
- **Documentation:** Luke Chung, Molly Pell, and Aparna Prophale
- **Quality Assurance/Support:** John Litchfield, Molly Pell, and Madhuja Vasudevan

Table of Contents

Table of Contents

Table of Contents

Welcome to Total Visual CodeTools!

Thank you for selecting Total Visual CodeTools for VB, Access/Office, and VBA. Total Visual CodeTools supports VB/VBA developers regardless of platform, so whether you are in Access, VB, Excel, Word, Visio, or any other VBA host, our coding tools are available to make you more productive.

Total Visual CodeTools is developed by FMS, the world's leading developer of products for Microsoft Access and Visual Basic. In addition to Total Visual CodeTools, we offer a wide range of products for Microsoft Access and VB developers, administrators, and users:

- EzUpData (cloud hosting of Access reports and files)
- Total Access Analyzer (database documentation)
- Total Access Admin (database maintenance control)
- Total Access Components (ActiveX controls)
- Total Access Detective (difference detector)
- Total Access Emailer (email blaster)
- Total Access Memo (rich text format memo fields)
- Total Access Speller (spell checker)
- Total Access Statistics (statistical analysis program)
- Total Access Startup (version launcher)
- Total VB Statistics (statistical analysis program)
- Total Visual Agent (database maintenance and scheduling)
- Total Visual SourceBook (code library)

We also offer products and services for SQL Server, Visual Studio .NET, and data analysts. Visit our web sites, www.fmsinc.com, and www.fmsasg.com, for more information. If you didn't purchase directly from us, please register online at www.fmsinc.com/support, and sign up for our free email newsletter. This guarantees that you are contacted in the event of news, upgrades, and beta invitations. Once again, thank you for selecting Total Visual CodeTools.

Luke Chung
President

Chapter 1: Introduction

Total Visual CodeTools provides a robust set of tools to help you with the day-to-day development of Visual Basic (VB) and Visual Basic for Applications (VBA) code. Covering everything from cleaning up inherited code to building the pieces of a robust and maintainable application, this program will become a part of your daily development efforts.

Topics in this Chapter

- **About Total Visual CodeTools**
- **Product Highlights**
- **Enhancements in This Version**
- **Enhancements in Previous Versions**
- **Visit Our Web Site**

About Total Visual CodeTools

Total Visual CodeTools is a collection of tools and utilities that make it easier to:

- Write solid code
- Reduce the drudgery of common coding tasks
- Create readable and maintainable code
- Clean up inherited code and standardize existing code
- Add sophisticated error handling and line numbering
- Find unused variables
- Create deliverable applications with code that is difficult to decipher
- Add line numbers to your code so you can pinpoint exactly which line crashed

To understand how Total Visual CodeTools works, it is important to view the program as a *collection* of tools to use during various phases of the programming process, from a project's inception to its final delivery and ongoing maintenance.

Total Visual CodeTools provides the features you need to create robust applications from beginning to end. The tools are presented in a toolbar that appears at the top of your VB/VBA IDE (editor window).

Product Highlights

Total Visual CodeTools supports all existing VB/VBA hosts. Integrated directly into the Integrated Development Environment (IDE), Total Visual CodeTools gives you a rich set of enterprise-enabled coding tools.

With Total Visual CodeTools, you can:

Ease Development in all VB/VBA Hosts

Total Visual CodeTools is available from within Microsoft Visual Basic 6.0, Office 2010, Office 2007, Office 2003, Office XP (2002), and Office 2000.

Share Project or Company Standards with your Entire Team

With Total Visual CodeTools, you can manage development standards for single developers, project teams, and entire enterprises. Set up standards for error handling, commenting, naming conventions, spacing and indentation, and more, then tell your entire team to point to the Standards

file! Your code will be consistent and readable, no matter how many developers work on the project.

Use Builders to Create Consistent Code at the Click of a Button

Total Visual CodeTools includes several Code Builders that help you accomplish common coding tasks quickly, while adhering to your standards for comments and error handling. By automating these tasks, you can cut the time you spend on tedious processes, and focus on the bigger picture. Total Visual CodeTools includes the following Code Builders:

- New Procedure Builder
- New Property Builder
- Long Text/SQL Builder
- Recordset Builder
- Message Box Builder
- Select CaseBuilder
- Copy Control Code Builder
- Format Builder
- DateDiff Builder

Standardize and Add Error Handling to Existing Code

Total Visual CodeTools Code Cleanup makes code adhere to your standards by adding error handling, standardizing indentation and comments, applying variable naming conventions, removing line numbers, and more. Use these tools on prototype code and inherited projects to apply consistent standards quickly.

Deliver Line Numbered and Obfuscated Code

The Code Delivery feature lets you add line numbers so your error handler can pinpoint the exact line where a crash occurs, minimizing the difficulties of working with users to replicate errors.

Code Delivery also lets you obfuscate your code in situations where you need to deliver source code, but don't want users to reverse engineer or modify it easily.

Identify Unused Variables

The Unused Variable Analysis tool identifies unused variables, classes, types, and elements in the selected procedure or module, or across all your project code. Cleanup your code and investigate potential problems.

Use Development Tools & Utilities to Increase Productivity

Total Visual CodeTools includes several development tools that help you complete your work faster. The development tools include:

- **Macro Recorder :** Records keystrokes to eliminate repetitive typing
- **Clear Immediate Window:** Empties the Debug Window
- **Close Code Windows:** Closes all open windows
- **Block Commenter:** Comments out text, and document the date, time, and developer in a comment line
- **VBE Color Schemes**: Displays a user interface to customize code editor fonts and colors

Enhancements in This Version

This version of Total Visual CodeTools is the seventh version of the product since the debut of Total Access CodeTools in 1996. The latest version introduces the following new features:

Supports Microsoft Office 2010, 32 bit Version

Total Visual CodeTools 2010 supports the 32 bit version of Microsoft Office 2010. It also supports Office 2000 through 2007, Visual Basic 6, and Windows 7 and earlier operating systems.

It does not support Office 2010, 64 bit version due to the complete change of the VBA IDE add-in integration. A future version of Total Visual CodeTools will support the 64 bit version.

VBA Code Parsing Supports Access/Office 2010

The VBA code parser that underlies the Code Cleanup, Code Delivery, and Unused Variable Analysis supports the new code and syntax in Office 2010 including the use of conditional compilers (e.g. #IF.. #ELSE.. #ENDIF) that is common for developers supporting both 32 and 64 bit environments. For situations where the same variable is defined more than once based on the conditional compiler directives, the first definition is used (for Code Cleanup and Delivery).

Code Cleanup and Code Delivery Allow Immediate Overwrite

After Total Visual CodeTools performs the analysis to cleanup or deliver your code, it prompts you to review the results before applying the changes. There is a new feature to let you bypass this step if no serious

errors were encountered. This eliminates this step so the entire process is completed without intervention. It is especially useful if you've successfully performed this task on a project before.

Code Cleanup Error Enabler and Handler Tags are Customizable

Total Visual CodeTools offers a feature to let you specify comment tags before and after each procedure's error enabler and error handler so you can easily replace them if you decide to change your error management code.

Previously, the comment tags for these were hardcoded, for example:

```
'TVCodeTools ErrorEnablerStart
'TVCodeTools ErrorEnablerEnd
```

Now, you can specify these values under Standards, Error Handling.

Copy Control Builder Supports Multiple Target Controls

You can now select multiple target controls and automatically generate the code you want to copy to all of them at one time.

Long Text/SQL Builder Converts Tabs and Spaces

The Long Text/SQL Builder lets you retrieve the SQL string from the query of a database you specify. It also offers new options to simplify the conversion of SQL queries and text blocks to VB6/VBA code. Tabs can be automatically replaced by spaces, and extra spaces can be eliminated.

Recordset Builder

The Recordset Builder has several enhancements:

- If opened from Access and referring to itself, when it's opened from another Access database, it'll default to itself
- New field selection buttons: Select All and Clear All
- Adds brackets around field names with special characters in them
- For Access databases, it understands field data types, and assigns appropriate values for text, numeric and date fields.

Message Box Builder

The Message Box Builder has a new user interface that supports Windows 7. The system modal option is removed because a message box is always modal, and options for right justification and right-to-left read are added.

Select Case Builder Supports Text Blocks and Numeric Ranges

The Select Case Builder now makes it easy to specify multiple case and variable assignment values. Simply paste a block of text with the values you want and the code is generated. You can also generate a numerical range of case values by specifying the start, end, and step values.

Total Visual CodeTools Menu Location Options

By default, the Total Visual CodeTools menu appears on the main menu bar. Under Standards, you can change this so it resides under the Add-Ins or Tools menu instead.

Tools Available During Debugging

Certain tools such as Clear Immediate Window and help are available when you are debugging your code. Previously, all the tools were disabled.

Default Send To is Remembered

The setting for the default Send To option is no longer under Standards. Total Visual CodeTools automatically saves the last option you select for the next time you open a builder.

Screens are Resizable

The forms for the builders and standards screen are now resizable to maximize their display on your screens.

Redesigned Storage of Standards

Rather than a text file, the standards are now stored in an Access database. This eliminates some of the challenges supporting international languages and double byte characters. If you install Total Visual CodeTools 2010 on a machine which had Total Visual CodeTools 2007, the previous settings are automatically imported. Otherwise, use the [Import Legacy CTS File] button from the Standards options under Manage Settings.

Enhancements in Previous Versions

Total Visual CodeTools 2007, Version 12 (March 2008)

Total Visual CodeTools 2007 introduces support for Microsoft Office 2007 with a complete overhaul and many new features:

New Tools and General Enhancements

- Unused Variable Analysis identifies unused variables, classes, types, and elements in the selected procedure or module, or across all project code
- Support for new Access/Office 2007 VBA syntax in Code Cleanup, Code Delivery, and Unused Variable Analysis
- Revised Standards section gives more control to individual builders. There is a new error handling template for forms/reports, ability to add new data types for naming conventions, and Recordset Builder variables are consolidated under naming conventions.
- Enlarged forms to better use higher resolution screens
- A more modern user interface including support for Windows XP Themes and manifest files in Visual Basic 6.0
- Revised user manual and help file with Vista support

Enhanced Builders

- For greater flexibility, the New Procedure Builder has options to add comments and error handling, which were previously set under Standards. There's an additional error handling template to handle form/report modules differently, and a Friend option for scope.
- The New Property Builder adds options to include or exclude comments and error handling.
- The Long Text/SQL Builder adds options for word wrapping SQL text on AND/OR and JOIN syntax.
- The Recordset Builder supports the Access 2007 ACCDB database format via ADO and DAO; includes the option to use the Access database and project objects; improves support for selecting the current and shared databases; supports creating a recordset for browse, add, or edit; and has options to use the field collections and setting query parameter properties. The ADO and DAO variable names use the centralized naming conventions and not another list.
- The Message Box Builder has an easier selection of icons rather than drag and drop, and reorganized options for the return variable.
- The Select Case Builder lets you assign a variable for each case value, and automatically assign it a value with automatic incremental assignment for numbers. There are options to place the variable on the same line and aligned with each other.

- The Copy Control Code Builder adds a Generated Text section to preview the new code before adding it.
- The Format Builder is organized into tabs and to allow adding and saving your own formats.
- Several builders automatically generate new code as you change options rather than requiring you to press a Build button.

Enhanced Other Tools

- Macro Recorder captures control key combinations, including clipboard operations. You can also record keystrokes in one module and run them in another.
- Close Code Windows adds an option to close only code or object windows in Visual Basic.
- Block Commenter goes way beyond comment and uncomment with options to put the block of code in IF FALSE blocks, #If..#End If compiler directive blocks, adding your comment text, and removing indentations.

Enhanced Code Cleanup

- Ability add additional data types to the Variable Naming Conventions list under Standards
- Option to exclude variables less than a certain length from the Variable Naming Convention cleanup
- When selecting options, new Select All and Clear All buttons

Enhanced Code Cleanup and Code Delivery

- Improved viewing and searching of Code Cleanup and Delivery preview
- Messages, including alerts, are placed in an HTML report
- Option to permanently hide the initial warning screen

Total Visual CodeTools 2003, Version 11 (December 2004)

Total Visual CodeTools 2003 includes the following new features:

- Support for Access/Office 2003 hosts in addition to Office 2000, Office XP, and Visual Basic 6.0.
- Three new Code Builders : Copy Control Code Builder, Format Builder, and DateDiff Builder.

- Macro Recorder Utility to record your keystrokes to help you increase your productivity.
- New Code Cleanup and Delivery options to "Save Form and Report Changes as they are Applied." This reduces memory requirements, and eliminates conflicts among objects.

Total Visual CodeTools 2002, Version 10 (January 2002)

Total Visual CodeTools 2003 includes the following new features:

- Support for Access/Office XP (2002) hosts in addition to Office 2000, and Visual Basic 6.0.
- Various enhancements to Code Builders to make them more useful during development.

Total Visual CodeTools 2000, Version 9 (January 2002)

Total Visual CodeTools 2000 is the first product in the Total Visual CodeTools line. Based on Total Access CodeTools, it combines support for all VB/VBA developers into one product. It introduces many new features, including:

- Support for Access/Office XP (2000) hosts and Visual Basic 6.0
- Centralized standards across Total Visual CodeTools tool
- Several new Code Builders, including the Recordset Builder, New Procedure Builder, New Property Builder, and Long Text/SQL Builder
- Many new enhancements for Code Cleanup to allow you to standardize your code.
- New Code Delivery feature to allow you to add line numbering and obfuscate code before delivery.

Visit Our Web Site

FMS is constantly developing new and better developer solutions. Total Visual CodeTools is part of our complete line of products designed specifically for the Access developer. Please take a moment to visit us online at **www.fmsinc.com** to find out about new products and updates.

Product Announcements and Press Releases

Read the latest information on new products, new versions, and future products. Press releases are available the same day they are sent to the

press. Sign up in our Feedback section to have press releases automatically sent to you via email.

Product Descriptions and Demos

Detailed descriptions for all of our products are available. Each product has its own page with information about features and capabilities. Demo versions for most of our products are also available.

Product Registration

Register your copy of Total Visual CodeTools on-line. Be sure to select the email notification option so you can be contacted when updates are available or news is released. You must be registered to receive technical support.

Product Updates

FMS is committed to quality software. When we find problems in our products, we fix them and post the new builds on our web site. Check our Product Updates page in the Technical Support area for the latest build.

Technical Papers, Tips and Tricks

FMS personnel often speak at conferences and write magazine articles, papers, and books. Copies and portions of this information are available to you online. Learn about our latest ideas and tricks for developing more effectively.

Social Media

Join us on Facebook: http://www.facebook.com/fms.solutions

And follow us on Twitter: http://www.twitter.com/fmsinc

Newsgroups

Share your experiences, learn from others, and ask your questions in our virtual community. Visit our newsgroups at:

```
www.fmsinc.com/support/newsgrp.htm
```

Or see our web site for additional instructions.

Links to Other Development Sites

Jump to other locations, including newsgroups, user group home pages, and other sites with news, techniques, and related services.

Chapter 2: Installation and Startup

Total Visual CodeTools comes with an automated setup program to get you up and running as quickly as possible. This chapter describes the system requirements, installation steps, instructions for upgrading from previous versions, and instructions for uninstalling. It also provides general guidelines and startup instructions.

Topics in this Chapter

- **System Requirements**
- **Upgrading from Previous Versions**
- **Installing Total Visual CodeTools**
- **Using the Update Wizard**
- **General Guidelines**
- **Launching Total Visual CodeTools**
- **Uninstalling Total Visual CodeTools**

System Requirements

Total Visual CodeTools has the following system requirements:

- VBA host (such as Office 2010, 2007, 2003, XP/2002, or 2000) or Visual Basic 6.0
- Operating system, processor, and memory that can run VB6 or a VBA host successfully
- 16 MB of free disk space to install the product (additional disk space may be necessary on a temporary basis while running the program, depending on the complexity of your code).

Upgrading from Previous Versions

If you have Total Visual CodeTools 2007 installed, follow these steps to upgrade without losing your Standards settings and program defaults:

1. Make a backup copy of your Standards file (*.cts). This can be found under Standards, Manage Settings. You can import the settings into Total Visual CodeTools, so that you do not need to reconfigure your settings.
2. Follow the steps in **Installing Total Visual CodeTools** below to install the program.
3. When Total Visual CodeTools 2010 runs for the first time, it automatically loads the settings from the 2007 version, if it can find the CTS file. If it can't find it, you can still import it into the new version under Standards, Manage Settings, Import Legacy CTS file.

After installation in complete, you can safely uninstall or remove all old versions of Total Visual CodeTools. Please follow the instructions on page 21 for uninstalling the program.

Installing Total Visual CodeTools

Total Visual CodeTools is installed using an automated setup program. To install Total Visual CodeTools, follow these steps:

1. If you have a previous version of Total Visual CodeTools installed, refer to **Upgrading from Previous Versions** above for information about upgrading without losing your Standards settings.
2. Run the setup program.
3. When prompted, enter your registration information and product key (serial number).
4. Specify the destination directory for the files.
5. Be sure to read the README file for any late breaking news that is not included in this User Guide.

Using the Update Wizard

If you are registered, you should receive emails from FMS when updates are released for the products. To verify you have the latest build, you can use the Total Visual CodeTools Update Wizard with an active Internet connection.

From the main Windows menu, select All Programs, FMS, Total Visual CodeTools, Update Wizard. Follow the prompts on the form to check for the latest update.

General Guidelines

Before using Total Visual CodeTools on your application, read this section for some basic guidelines for using the product.

Make Regular Backups

Your project represents a far greater amount of time and effort than most other computer files; it is not unusual to spend many months of development on one project. Because of this, you should make frequent backups of you database files.

Your projects represent a huge amount of work—protect them accordingly.

If you plan to use Code Cleanup or Code Delivery on your project, you should always make a backup before running the program. See page 126 for more information about the importance of backups for the Code Cleanup and Code Delivery tools.

Permissions and Security

For Total Visual CodeTools to interact with and/or modify your project, it needs to have adequate permission levels to view and change objects. This means that you need to be logged into your application development environment with read/write permissions before you start Total Visual CodeTools.

- For Microsoft Access, ensure you are logged into an account that has full permissions for forms, reports, modules and classes.
- For Office applications, including Access, ensure that you entered your VBA Project Password (if one was previously set). To do this, open the module editor and select Tools|Project Properties. On the Protection tab, enter your password to unlock the project for viewing.
- For Visual Basic projects, ensure that you have full file and directory rights for all objects and locations that represent files in your projects. If one or more parts of your project are stored on network drives, ensure that those network drives are available before running Total Visual CodeTools.

Avoid Using Shared Projects

The Code Cleanup and Code Delivery features of Total Visual CodeTools can make extensive changes to your code. You should never run these features on a shared project.

- For Microsoft Access/Office projects, be sure that you are the only developer using the project before running Code Cleanup or Code Delivery. Of course, before running either tool you should have made a backup of your project and should be running the backup version. This eliminates the threat of multi-user issues.
- For Visual Basic applications, the only real way to share projects is to use Microsoft Visual SourceSafe. If you are using SourceSafe, read **Issues with Visual SourceSafe** below.

Issues with Visual SourceSafe

Visual Basic and Office hosts integrate with Microsoft Visual SourceSafe to allow multiple developers to work on a project with version control and check in/check out facilities. You cannot save changes to objects that are under Visual SourceSafe control unless you have the object checked out.

Because Code Cleanup and Code Delivery can modify your code, check out all your modules before using these features.

Compact and Repair Access Databases

If you are using Access, database corruption can occur. Before using Total Visual CodeTools (or any add-in program) on your database, be sure that your database is in its most efficient form by compacting and repairing it.

To do this for Access 2007 or Access 2010, open your database, then select from the Office Button, Manage Database, Compact and Repair Database. For earlier versions of Access, select from the Access menu *Tools|Database Utilities|Compact and Repair Database*.

For optimal performance, your databases should be regularly compacted and backed up. If you do not have an automated process in place, consider **Total Visual Agent** from FMS, which automates these administrative chores. For more information and a demo, visit www.fmsinc.com.

Ensure Adequate Disk Space for Code Cleanup and Delivery

When you run the Code Cleanup or Delivery tools, Total Visual CodeTools runs through a complex set of operations to read and parse your module code. This can take a fair amount of temporary disk space. Verify that there is enough free disk space on your local drive before running these tasks. A good rule of thumb is to have free space equal to at least 3 times the size of the database or project with which you are working.

Launching Total Visual CodeTools

Total Visual CodeTools runs from within the Visual Basic Integrated Development Environment (IDE) or the VBA IDE. If you are familiar with VB/VBA editor, you should be very comfortable using Total Visual CodeTools.

Available from the Add-Ins Menu

When you install Total Visual CodeTools, the setup program creates registry entries so the Microsoft Visual Basic and Office programs can integrate it into their add-ins menu. Total Visual CodeTools installs as a “loaded” add-in. This means that whenever you start Visual Basic or the Office Visual Basic Editor, the CodeTools menu item and toolbar is ready to go.

Because of Microsoft's support for VB6/VBA add-ins on a user rather than machine basis, if you installed Total Visual CodeTools under a different Windows login name, you'll need to load it again. For Visual Basic, you can select it from the Add-ins Menu. For VBA, you'll need to update the registry.

For your convenience, a REG file is included in the installation directory that you simply need to run. Locate the file named "VBA-Addin.reg", and double click to execute this. A registry warning or Windows security message may appear. For Vista and Windows 7 operating systems, you may need admin rights to make the registry change.

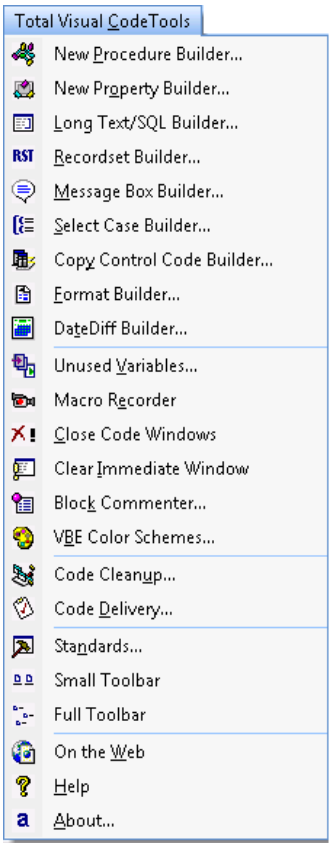
This registry file contains the following information:

```
[HKEY_CURRENT_USER\Software\Microsoft\VBA\VBE\6.0\Addins\TV
CTLVBA.CAddInConnect]

@=""
"FriendlyName"="Total Visual CodeTools 2010"
"LoadBehavior"=dword:00000001
"Description"="Total Visual CodeTools 2010 from FMS, Inc."
```

Running Total Visual CodeTools

When Total Visual CodeTools is loaded, the menu item Total Visual CodeTools is added to your IDE menu and a toolbar is available.

The location of the menu can be changed to appear under the Add-Ins or Tools menu under the Standards options.



Total Visual CodeTools Menu

From the Total Visual CodeTools menu item, you can select any of the available tools:

Total Visual CodeTools Toolbars

The Total Visual CodeTools toolbars makes it easy to select its many tools. The toolbar behaves like other VB/VBA toolbars and can float over your workspace or be docked.

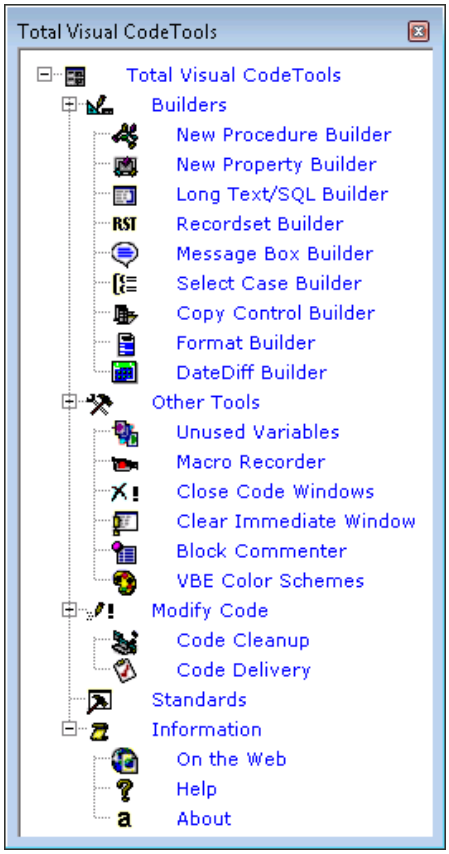
Choose from two views of the toolbar: Small or Full.

Small Toolbar

Small Toolbar

The Small Toolbar shows tools in a compact mode. This view is well suited for development work when space is a concern. You can also dock this to the edge of your IDE.

As your mouse hovers over each button, a tip appears explaining the tool. If you close the toolbar and want to retrieve it, select Small Toolbar from the Total Visual CodeTools menu, or right click on a code window and choose Total Visual CodeTools, Small Toolbar.



Full Toolbar

The Full toolbar shows all available tools organized into categories with their full names for easy selection. This toolbar is useful at high resolutions where space is not a constraint.

To change to the Full Toolbar view from the Small Toolbar view, select Full Toolbar from the Total Visual CodeTools menu, or right click on a code window and choose Total Visual CodeTools, Full Toolbar.

Setting Options and Standards

One of the first steps in using Total Visual CodeTools is to configure the program to use the options you want, and the standards you use for coding. To do this, press the [Standards] button and edit the fields that you want to customize. Refer to **Chapter 3: Managing Standards** on page 23 for more information.

Context Sensitive Help

Throughout Total Visual CodeTools, you can press the [Help] button or [F1] for context-sensitive help. The online help file is fully indexed, and may include more up-to-date information than the printed user manual.

Uninstalling Total Visual CodeTools

Total Visual CodeTools conforms to Windows installation and removal standards. To uninstall Total Visual CodeTools:

1. From the main Windows menu, select Control Panel.
2. From the Control Panel window, select Programs and Features (or Add/Remove Programs for Windows XP or earlier).
3. Select Total Visual CodeTools 2010 from the list of installed programs, and click the Change/Remove button.
4. Follow the onscreen prompts to uninstall the product.

Note that the uninstall process does not remove any files created by Total Visual CodeTools after it was installed. For example, a temporary database may still exist in the Total Visual CodeTools folder or working folder under your Users profile. After uninstalling the product, check these folders for any remaining files that need to be manually deleted.

Chapter 3: Managing Standards

Each tool in Total Visual CodeTools has its set of options, and shares some settings with other tools. These settings are specified under Standards. This chapter explains how to customize the Standards to your coding style. These standards can also be specified and enforced across your entire development team so everyone is using the same commenting and error handling structure, formatting style, variable naming conventions, etc.

Topics in this Chapter

- **Introduction**
- **Standards Architecture**
- **Setting Standards**
- **Managing Settings Files**
- **Shared Settings Scenarios**
- **Settings Cross-Reference**

Introduction

One of the most important features of Total Visual CodeTools is its ability to manage development standards for single developers, project teams, and entire enterprises. As projects become more complex and entail greater staffing requirements, the need for standards in development becomes crucial.

Total Visual CodeTools handles this need by offering a powerful and flexible architecture for managing your standards.

- As a single developer, you can use Total Visual CodeTools to ensure that your entire project conforms to the development standards dictated. With support for multiple standards templates, you can customize standards according to a specific client's needs, or a particular project's requirements.
- As a team or enterprise developer, you can use Total Visual CodeTools to develop projects without worrying about whether or not you are conforming to the defined project standard.
- As a development manager, you now have a tool to define development standards, and enforce them across entire project teams.

How to Use this Chapter

This chapter describes the standard architecture and all the settings in detail. If you are not concerned about sharing your standards with others, and want to get up and running quickly, you may want to jump ahead to the following chapters that describe each tool. Once you've identified a tool you're interested in, refer to the sections of this chapter that are relevant.

If you're immediately interested in using the Code Cleanup feature which modifies all your code, you should read this section because many of the settings impact Code Cleanup.

Standards Architecture

Total Visual CodeTools uses Standards to define how your projects are created and updated. Standards define naming conventions, commenting style, error handling, etc. In order to use Standards effectively, you should understand the Standards architecture employed in the product.

Types of Settings

Total Visual CodeTools uses two types of settings to control how the product works and how standards are enforced.

Local Settings

Local Settings are the options for a specific tool that are not shared with other developers. For instance, on the New Procedure Builder whether you want to add error handling or not.

Another example is the developer's initials, which can be used when generating comments. Local Settings are stored in the users section of the Windows Registry, and are therefore private to the developer.

Shared Standards

Shared Standards are more global standards that are used throughout the Total Visual CodeTools program. These specify development standards that can be shared by an entire team. These standards are stored in a file that can be located on a network drive and shared across multiple developers using Total Visual CodeTools.

For example, naming conventions, commenting style, and error handling settings are Shared Standards. All Shared Standards are set and managed from the Total Visual CodeTools Standards form. This enterprise feature is extremely helpful for maintaining consistent coding styles across your development team, and is particularly effective for new members of your team or junior programmers.

For a complete list of Shared vs. Local settings, see **Settings Cross-Reference** on page 64.

Shared Standards File

Total Visual CodeTools stores all Shared Standards in one or more Shared Standards databases. When you first install the product, a default file is created in your user folder. This file contains reasonable defaults—if you are a single developer and want to get started right away, the default settings will most likely be appropriate.

You can modify any Shared Standards to match your development style and requirements. Additionally, you can save multiple settings in multiple Shared Standards files. This is helpful if you are working on multiple projects with different development standards.

Finally, you can place a Shared Standards file on a shared network drive and have every developer's installation of Total Visual CodeTools use this file. The settings can be password protected, so you can define and protect standards from unwarranted modification.

Working in Laptop/Disconnected Environments

Having all developers working with Total Visual CodeTools use a shared settings file is a great concept, but what happens when a developer disconnects from the network? Developers often disconnect to work offsite—what happens to the connection to the settings file in such a case?

A local copy of the shared standards file is made so that if the network version is not available, the last set of standards is still available. Total Visual CodeTools automatically checks at startup to see if its connection to the Shared Settings file is available. If not, it switches to a local copy of the settings. This means that Total Visual CodeTools can be used even when disconnected from a Shared Settings file located on a shared drive.

The next time the disconnected developer connects to the shared drive and starts Total Visual CodeTools, the program automatically reconnects to the Shared Settings file on the shared drive.

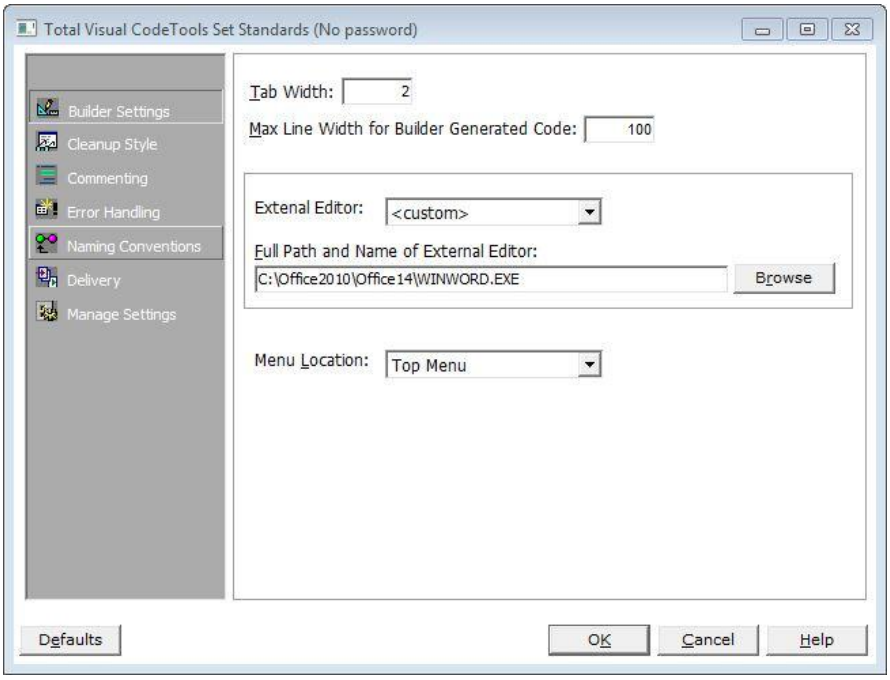
Note that if you have assigned a settings file to the Shared Settings file, and a developer is working in disconnected mode, the developer cannot change any settings—this is the design of password protection. However, the developer may need to change certain settings in response to project requirements. In such a case, the developer can clear the password on his or her local copy using the Save As feature.

See the **Managing Settings Files** section on page 60 for more information.

Setting Standards

The Standards screen is invoked from the main menu under Standards, or the Standards button on most builders where it's relevant.

The Standards form appears:



Standards Form: Builder Settings

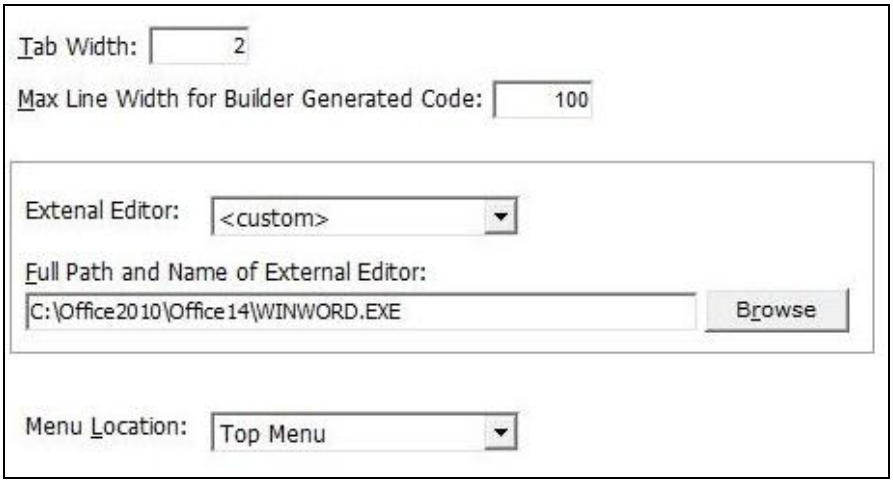
The Standards form is divided into these sections, with its menu items on the left side:

- Builder Settings
- Cleanup Style
- Commenting
- Error Handling
- Naming Conventions
- Delivery
- Manage Settings

Selecting these options brings up the relevant settings on the right side. Simply make the changes you want and press the [OK] button to save them. The settings are automatically saved in the file name specified under Manage Settings.

Builder Settings

The Builder Settings specify some basic options for how the code builders in Total Visual CodeTools behave:



Standards Form: Builder Settings

Tab Width

The Tab Width determines how far to indent lines. By default, it uses your VB/VBA setting when you first run Total Visual CodeTools. This setting is used by the builders when they create code for you, and also by Code Cleanup when it standardizes your code indentations (e.g. Sub..End Sub, If..End If, For..Next, etc.)

Max Line Width

Some of the code builders, like the Long Text Builder, can generate lines that are quite lengthy. You have the option to force "word-wrapping" by specifying the maximum length to allow on a line. When the text approaches this limit, a " _" is added and remaining text wrapped to the next line.

External Editor

The Code Builders give you options to send the code to the project, clipboard, or file. You can also specify an external editor like Notepad. You can provide an alternative editor by choosing <custom> and specifying the editor.

Menu Location

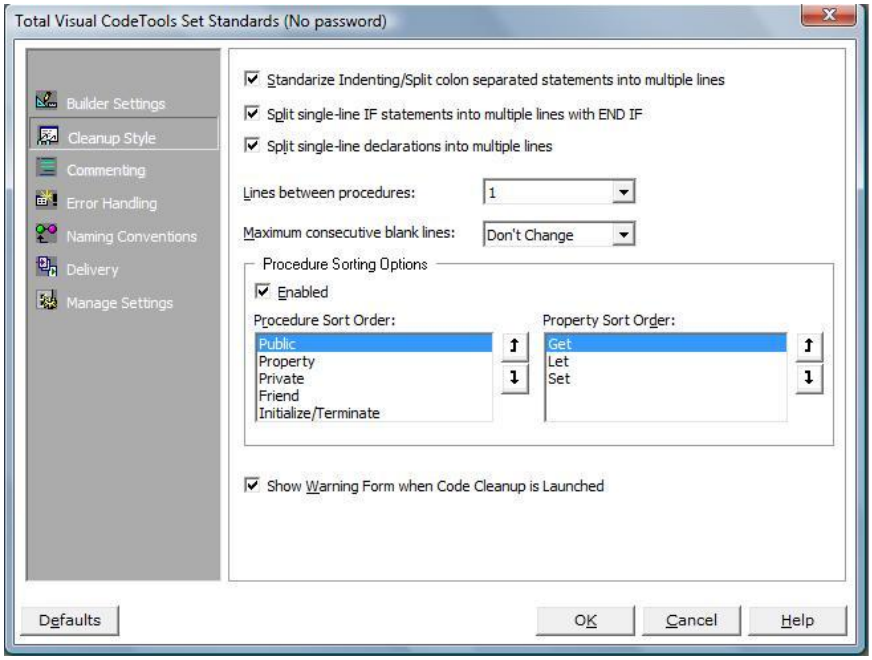
You can specify where you want Total Visual CodeTools to appear on your IDE.

Menu Location Options

It can appear on your main menu, under “Add-Ins” or under “Tools”. When you make and save the change, you need to exit your program entirely and restart it to see the change.

Cleanup Style

The Cleanup Style settings let you specify how you’d like the Code Cleanup feature of Total Visual CodeTools to behave:



Standards Form: Cleanup Style

Standardizing Code Formatting

It's nearly impossible to maintain someone else's non-indented or awkwardly indented code. Consider the following function:

```
Function TestVars(A As Integer, B As Integer) As Double
Dim dblReturn As Double
Dim x As Integer
dblReturn = 10
If A <= 20 Then
For x = 1 to 20
dblReturn = dblReturn + x ^ B
Next x
Else
x = A
Do
x = x + 1
dblReturn = dblReturn + x * B
Loop Until x > A
End If
TestVars = dblReturn
End Function
```

Now compare this code to the version with standardized indenting:

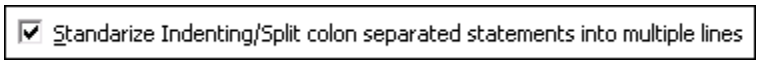
```
Function TestVars(A As Integer, B As Integer) As Double
  Dim dblReturn As Double
  Dim x As Integer
  dblReturn = 10
  If A <= 20 Then
    For x = 1 to 20
      dblReturn = dblReturn + x ^ B
    Next x
  Else
    x = A
    Do
      x = x + 1
      dblReturn = dblReturn + x * B
    Loop Until x > A
  End If
  TestVars = dblReturn
End Function
```

Which function is easier to support? Using the Code Cleanup tool, you can apply standardizes indenting to each line of your module code. The number of spaces for each indentation is defined by the Tab Width specified on the Builder Settings page.

Standardize Indenting/Split Colon

VB/VBA allows developers to add multiple commands (lines of code) on one line by separating them with colons. This makes it difficult to read and

understand, especially if unexpected code is buried in a line of other commands.



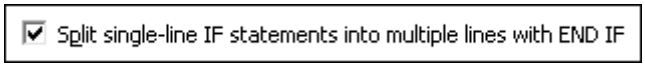
When you select the "Standardize Indenting/Split Colon" option, Code Cleanup indents your lines properly and splits any lines with multiple statements separated by colons. For example, the following code:

```
If fValid Then
Select Case strTable
Case "Customers": strIndex = "LastName"
Case "Orders": strIndex = "OrderID": fBigTable = True
Case "Employees": strIndex = "EmpID"
End Select
End If
```

Becomes:

```
If fValid Then
  Select Case strTable
    Case "Customers"
      strIndex = "LastName"
    Case "Orders"
      strIndex = "OrderID"
      fBigTable = True
    Case "Employees"
      strIndex = "EmpID"
  End Select
End If
```

Split Single-line IF Statements



VB/VBA allows developers to put If statements on one line, which can be very difficult to read and understand. The "Split Single Line If Statement" option separates such statements onto separate lines with a closing End If.

For example, the following code:

```
If x > 50 Then RunMyProc Else Exit Do
```

Becomes:

```
If x > 50 Then
 RunMyProc
Else
  Exit Do
End If
```

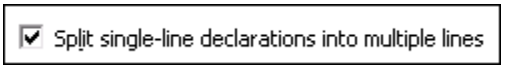
Colons can also be used in the THEN and ELSE clauses, which makes the original comment even more difficult to understand:

```
If fOK Then RunMyProc: RunMyProc2 Else RunMyProc3:
RunMyProc4
```

When you select the "Standardize Indenting/Split Colon" option along with the "Split Single-line If Statements" option, Code Cleanup converting the statement to the following:

```
If fOK Then
  RunMyProc
  RunMyProc2
Else
  RunMyProc3
  RunMyProc4
End If
```

Split Single-line Declarations



VB/VBA allows multiple variable declarations on one line, such as:

```
Dim X, Y, Z As Integer
```

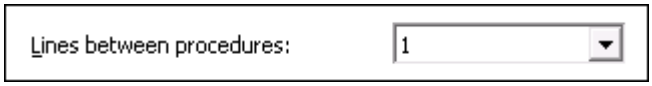
Although this is convenient, it makes your code more difficult to decipher, possibly leading to hidden bugs, and makes your code difficult to read (see page 165 for additional information).

Code Cleanup fixes this by splitting every Dim statement into its own line, leaving you with code that is easier to read and debug:

```
Dim X
Dim Y
Dim Z As Integer
```

In addition to DIM statements, other variable declarations such as Public and Private in declarations, and Static variables in procedures are also separated.

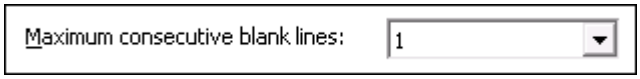
Lines between Procedures



Procedures can be separated with different numbers of lines or no lines at all. Code Cleanup can fix inconsistent spacing to a user specified number of lines between procedures and properties.

To enable the feature, select the desired number from the combo box. To disable the feature, select the “Don’t Change” option.

Maximum Consecutive Blank Lines

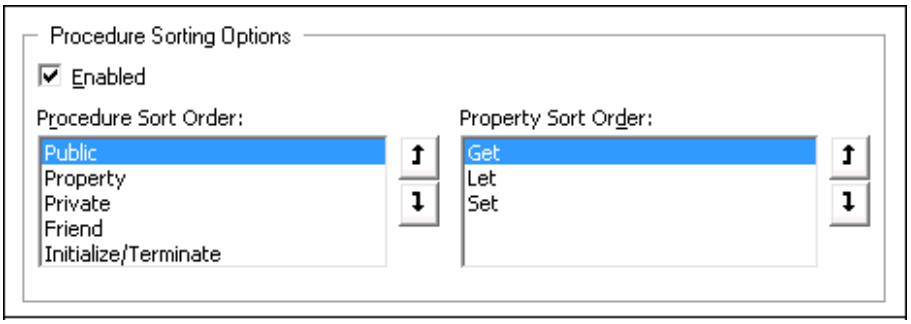


For consistency, you may not want more than a certain number of consecutive blank lines in your module code. Use this option to specify the maximum number of blank lines allowed in your code, or select the “Don’t Change” option to disable adjustments the feature.

Procedure Sorting

The VB/VBA editor lets you add properties and procedures in any order. This lets you see multiple procedures and keeps them in the exact order you entered them. While this behavior can be very helpful, you may also end up with procedures not sorted in optimal order.

The “Procedure Sorting” option allows you to group your procedures by type, and sort them alphabetically by procedure name:



Procedure Sorting Options

To enable this option, select the “Enabled” check box and use the up and down arrows to specify the procedure and property sort order.

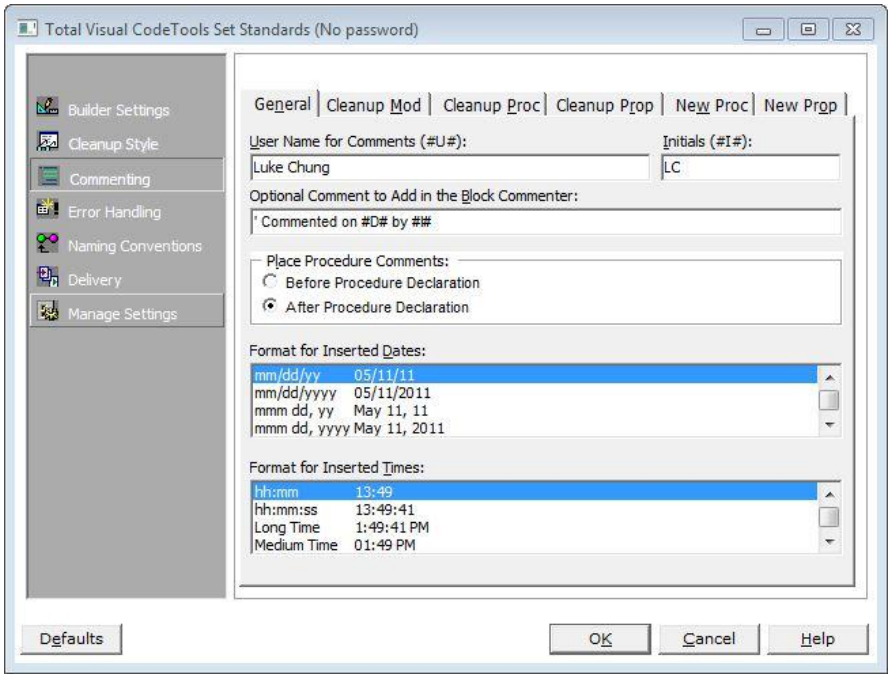
Show Warning

When the Code Cleanup feature is started, a warning message appears to confirm you’ve backed up your project. This option lets you hide that warning. If you hide it, you can turn it back on by checking this box:

Commenting

The Commenting section lets you specify the commenting structure you'd like to see with these tools:

- Code Cleanup when Total Visual CodeTools adds comments to existing modules and procedures that lack it
- New Procedure Builder when you create a new procedure
- New Property Builder when you create new property Let, Get, and Set statements



Standards Form: Commenting

The Commenting Standards are divided into five tabs:

General Tab

For general commenting options that span all comments

Cleanup Tabs for Module, Procedures, and Properties

Comment structures and options when Code Cleanup adds comments to existing modules, procedures and properties.

New Procedure

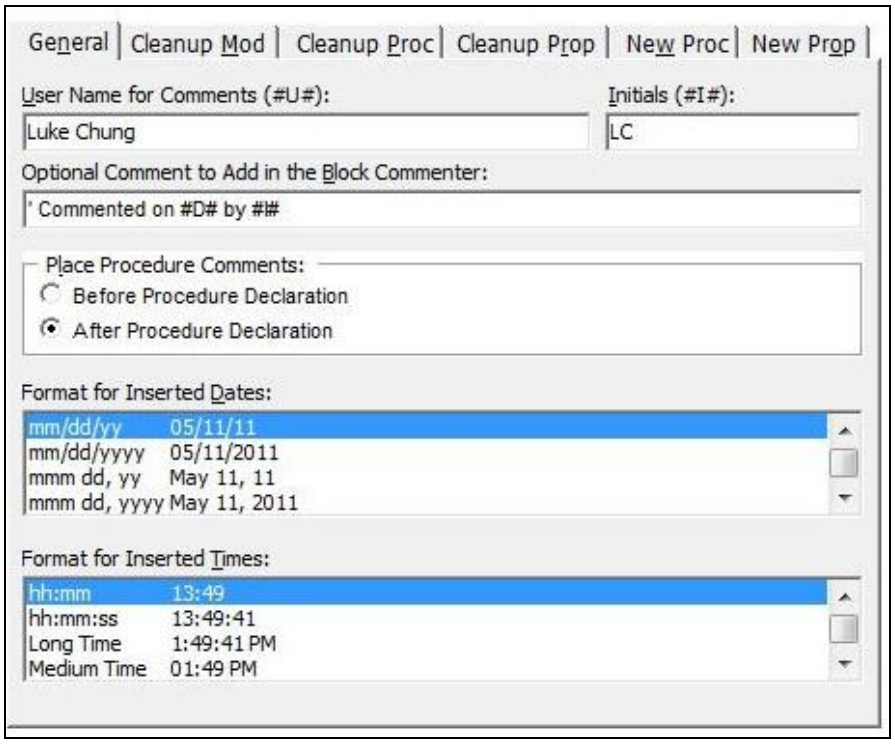
Comment structure for the New Procedure Builder when creating new procedures. This differs from the Cleanup Proc setting since you can add items like Create Date/Time.

New Property

Comment structure for the Property Builder when creating new properties. This differs from the Cleanup Prop setting so you can add items like Create Date/Time.

General Tab

This tab contains general options that apply to all comments:



Commenting Standards, General Tab

Use this tab to specify:

- Your name and initials, which can be referenced in comments
- Comments to add to the Block Commenter tool when that tool's Add Comments option is selected.
- Where Code Cleanup should place procedure comments

  - Before to place the comments above the procedure definition line (where Sub/Function exists)
  - After to place them below the procedure definition line
- The format for dates and times

The developer name and initials are stored in the registry and not the shared standards file, so each developer sets their own values.

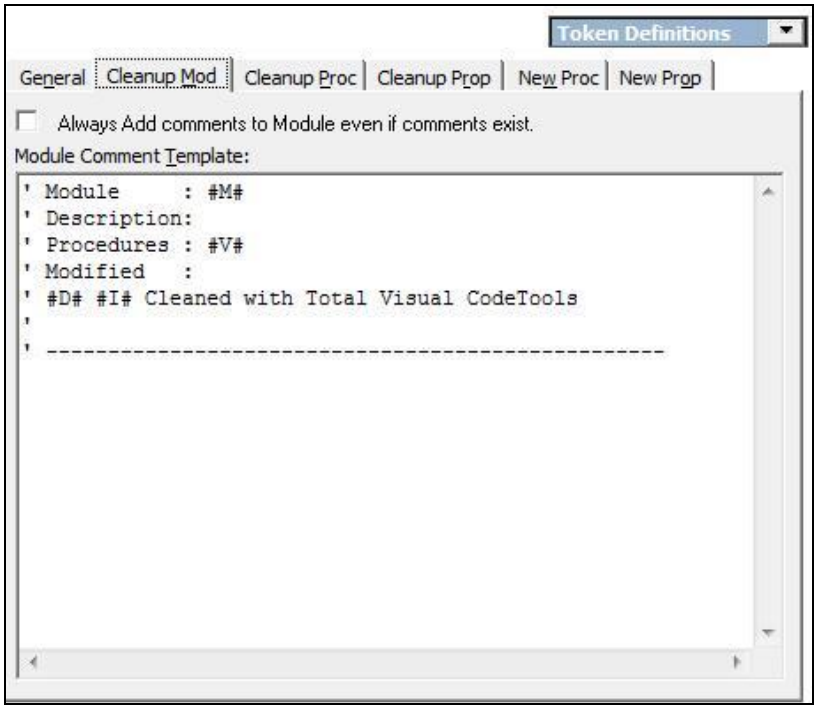
Notice the #D# and #I# text in the Block Commenter text box. These tokens are substituted when the comment is added. In this case, #D# is the current date and #I# is the developer's initials as specified by the Initials text box. More information on tokens is available under **Commenting Token Definitions** on page 38.

Code Cleanup: Module Comments (Cleanup Mod tab)

The Code Cleanup feature offers an option to add "Module Comments" which applies a standard comment block to the beginning of every module. Although the program cannot deduce what your code actually does, it can add the comment block to minimize the work of inserting it manually.

Total Visual CodeTools can also add a list of the procedures in the module with the complete procedure definition (parameters and return data type). This feature allows you to include an inventory of procedure names right in the module comments. See page 169 for more information about the importance of adding comments.

The format of the module comments is set on the Cleanup Mod tab of the Commenting page:

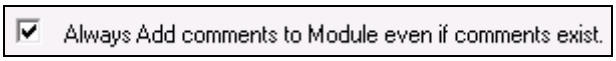


Commenting Standards, Cleanup Module Tab

Always Add Comments Option

By default, Code Cleanup does not add module comments if it detects existing module comments. This is determined by comparing the first module line with the first line in the module template. If they match, comments are not added again. This prevents the addition of module comments every time you run Code Cleanup.

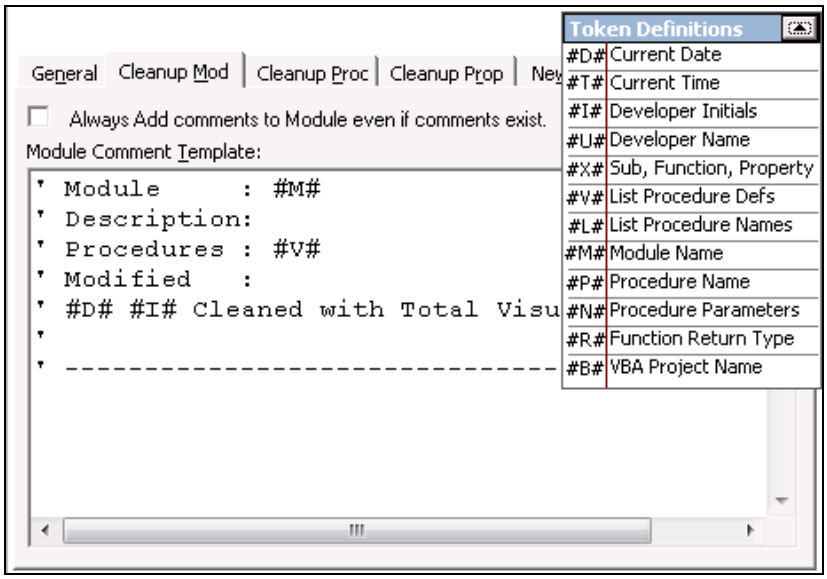
If you want to always add comments to every selected module, select the "Always Add Comments" option:



Module Comment Template

The bottom part of the Cleanup Module tab lets you customize the format of your module comment template.

In addition to regular text, Total Visual CodeTools includes a powerful feature to let you insert runtime values called **Tokens** into your comments with the *#Letter#* syntax.



Commenting Standards, Cleanup Module Tab with Tokens

When these characters are included in the comment template, Code Cleanup replaces them with the actual values when the code is written to your project. The Tokens dropdown list shows the available options, which can also be used in customizing error handling routines.

Commenting Token Definitions

These tokens are most relevant for module comments:

Token	Meaning
#M#	Module name
#Y#	List of procedure definitions (procedure names, parameters, and return values) in the module
#V#	List of procedure definitions (procedure names and parameters, without return value) in the module
#L#	List of procedure names (without parameters) in the module
#U#	Developer (user) name as specified on the General tab
#I#	Developer's initials as specified on the General tab
#D#	Current date formatted based on the General tab
#T#	Current time formatted based on the General tab

Sample Module Comments

These examples show you the results of using Total Visual CodeTools default module commenting features:

```
' Module      : Startup
' Description:
' Procedures  : CloseForm()
'               FixAllDataAccessPages()
'               FixPageConnection(pstrDBLocation As String,…
'               FormOpen()
'               HideStartupForm()
'               IsItAReplica()
'               IsItMDE(pDbs As Database)
'               MarkPagesUnfixed()
'               OpenStartup()
'               WaitForDAPToLoad(pobjDap As Object)
' Modified    :
' 06/05/11 MP Cleaned with Total Visual CodeTools
' ---------------------------------------------------------
```

If you use the procedure name (#L#) token instead of the procedure definition (#V#) token, the procedures section looks like this:

```
' Procedures  : CloseForm
'               FixAllDataAccessPages
'               FixPageConnection
'               FormOpen
'               HideStartupForm
'               IsItAReplica
'               IsItMDE
'               MarkPagesUnfixed
'               OpenStartup
'               WaitForDAPToLoad
```

At FMS, we use the module comment header to identify the creator of the module and all modifications to the module, by procedure. The module level comments summarize changes across the whole module but not at the detail level of comments stored in procedures. This is particularly useful for describing new features or changes that impact many procedures.

Module comments can also be used to add copyright information or other text that you want at the top of every module. To prevent it from getting deleted when the Code Delivery feature removes all comments, use the special comment preservation character (by default '!) before each line. See page 59 for more information.

Code Cleanup: Procedure and Property Comments (Cleanup Proc and Cleanup Prop tabs)

When you select the “Procedure Comments” or “Property Comments” options from the Code Cleanup form, Total Visual CodeTools applies a comment block to the beginning of every procedure or property.

For instance, a procedure with the definition:

```
Function Mystery(Param1 As String, Param2) As Boolean
```

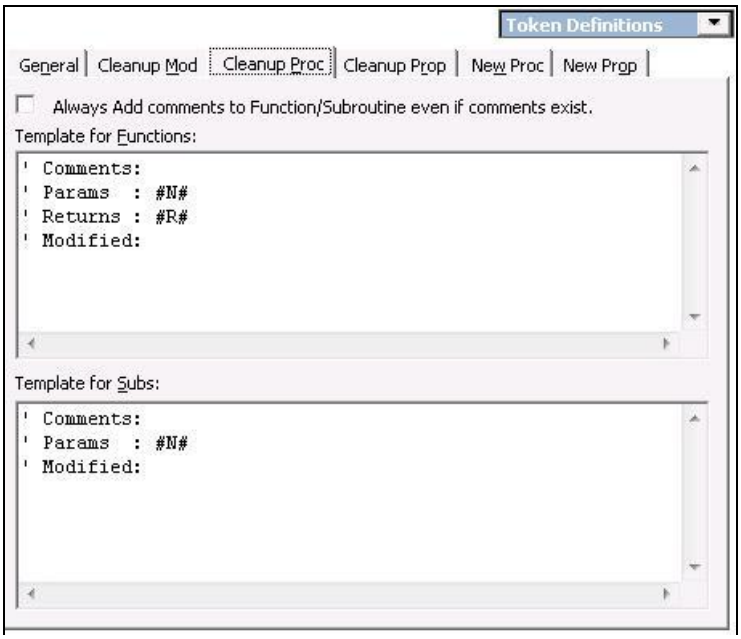
Gets this comment block added (if the After option is selected on the General tab):

```
Function Mystery(Param1 As String, Param2) As Boolean
  ' Comments:
  ' Params  : Param1
  '           Param2
  ' Returns : Boolean
  ' Modified:
```

Notice that each of the procedure’s parameters is listed along with the return type. The Modified section is intended for adding information whenever you modify the procedure. At FMS, we add a new line for each change with the date, developer initials and change description like this:

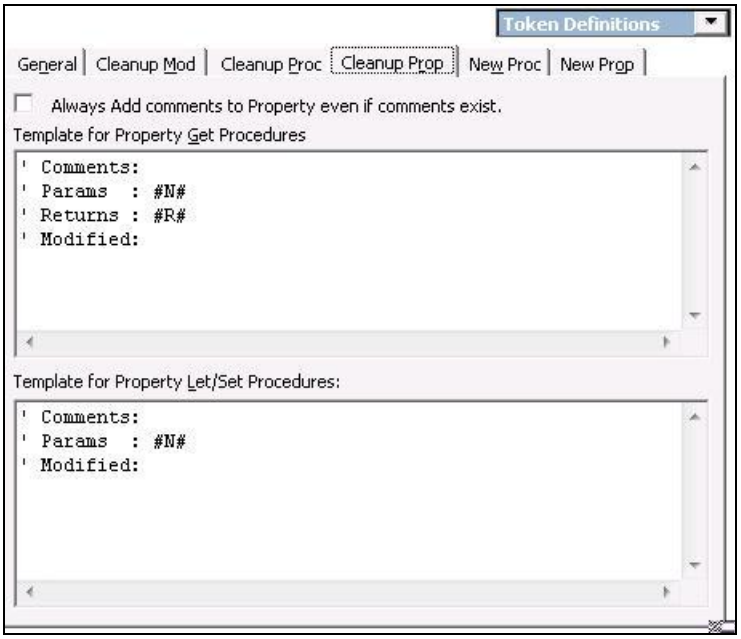
```
  ' Modified:
  ' 05/31/11-MP Fixed ...
```

The standard comment block for each procedure makes it easy to enter details, including the definition/purpose of each parameter, sparing you the work of repeatedly typing the block of text. Since the comments are completely customizable, you end up with code that is consistently commented according to your standards. Different formats can even be specified for Functions vs. Subs (to handle return value):



Commenting Standards, Cleanup Procedure Tab

Similarly, the Cleanup Prop tab allows separate entries for Get vs. Let/Set procedures:



Commenting Standards, Cleanup Property Tab

Comments are added before or after the procedure definition line as specified on the General tab.

Always Add Option

By default, Code Cleanup does not add procedure or property comments if it detects existing comments. This is determined by comparing the first module line with the first line in the module template (excluding the colon). If they match, comments are not added again. This prevents the addition of module comments every time you run Code Cleanup.

If you want to always add comments to every procedure, select the "Always Add Comments" option. This may be useful if you want to get the list of current parameters for each procedure.

Procedure/Property Comment Templates

The bottom part of the Cleanup Proc tab lets you customize the procedure comment templates for Functions and Subs, while the Cleanup Prop tab lets you customize the property comments for Get and Let/Set procedures.

Your comments should be flush left; do not indent the lines. If you place comments before the procedure definition (set on the General tab), the comments are added without indentation. If the comments are added after the procedure definition, they are indented one tab width as specified on the Builder Settings tab.

Procedure and Property Tokens

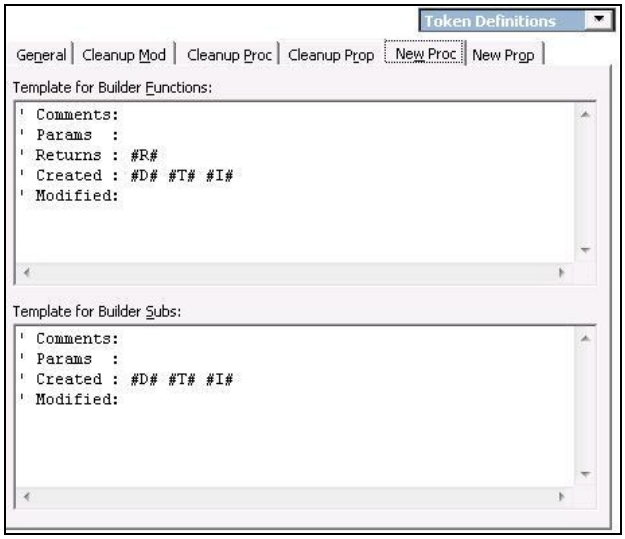
In addition to regular text, Total Visual CodeTools includes a powerful feature to let you insert runtime values called **Tokens** into your comments with the #*Letter*# syntax.

Click on the Tokens drop down list to see the available options. The following tokens are particularly useful for procedure comments:

Token	Meaning
#P#	Procedure name
#N#	List of procedure parameters
#R#	Data type of the function's return value
#U#	Developer (user) name as specified on the General tab
#I#	Developer's initials as specified on the General tab
#D#	Current date formatted based on the General tab
#T#	Current time formatted based on the General tab

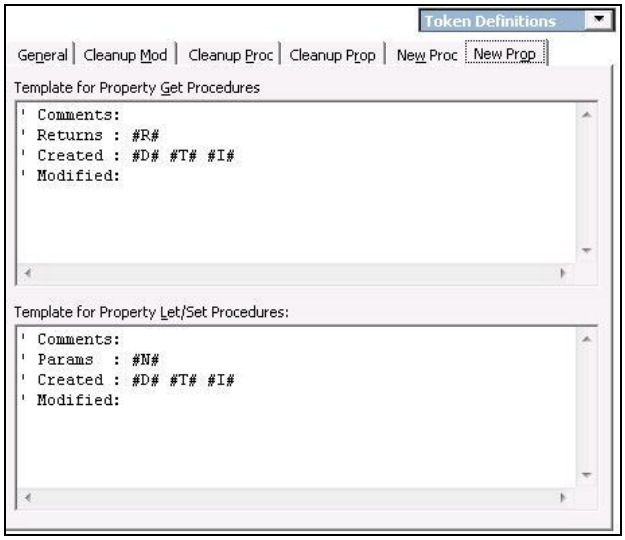
New Procedure and New Property Comment Tabs

The New Proc and New Prop tabs apply to the New Procedure and New Property Builders respectively. When the Add Comments option is selected on those builders, the comment block is added to the new procedure/property. Unlike the comments for procedures and properties in Code Cleanup, these comments can include the current date, time, and user initials so we can stamp when and who wrote it.



Commenting Standards, New Procedure Tab

The New Properties comments tab is similar:



Commenting Standards, New Property Tab

In addition to regular text, Total Visual CodeTools lets you replace the *#Letter#* tokens with runtime values. Click on the Tokens drop down list to see the available tokens. The tokens relevant to procedures and properties are listed under **Procedure and Property Tokens** on page 42.

Sample Procedure Comments

The default settings are close to what we use at FMS to manage our development projects (we don't tag the create time). For each procedure, we list a brief comment describing the procedure, the parameters and their purpose, the return value, when it was created, and the developer who created it. Additionally, we keep a log of every revision with the date, person, and modification made. For example, a procedure's comments might look like this:

```
Function GetID(strFirst As String, strLast As String) _
  As Integer
  ' Comments: Get customer ID based on the customer name
  ' Params  : strFirst   Customer's first name
  '           strLast    Customer's last name
  ' Returns : Customer ID
  ' Created : 06/01/11 LC
  ' Modified:
  ' 05/03/11 DH Added support for duplicate names
  ' 07/07/11 LC Used secondary key
```

Error Handling

The Error Handling section lets you specify the error handling structure you'd like to see with these tools:

- Code Cleanup when Total Visual CodeTools adds error handling to procedures that lack an On Error command
- New Procedure Builder when you create a new procedure
- New Property Builder when you create new property Let, Get, and Set statements

Every procedure, no matter how small or insignificant, should have some type of error handling. Without error handling, your application can leave users looking at cryptic error messages and leave data in an inconsistent state. At the most rudimentary level, there should be an On Error GoTo statement that points VB6/VBA to a label within the procedure. This label should contain code that, at a minimum, displays a meaningful error message.

For more information on the importance of error handling, see **Implement Robust Error Handling** on page 161.

For example, the following procedure tries to delete a file:

```
Function DeleteFile()
  Kill "C:\Myfile.txt"
End Function
```

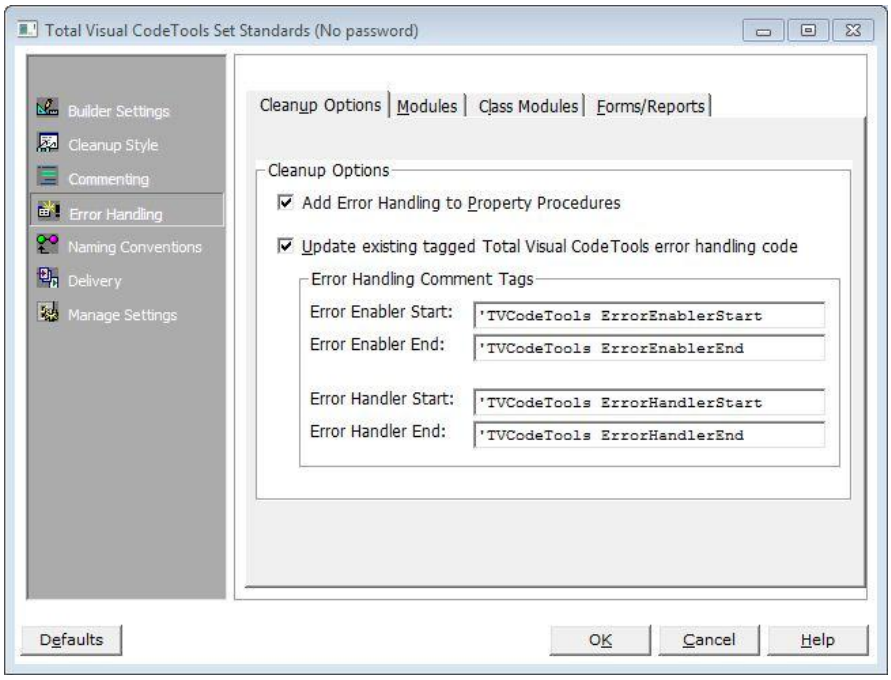
If the file does not exist, a runtime error occurs and presents your application's user with an Access error message dialog.

If you select the "Error Handling" option from the main Code Cleanup form, Total Visual CodeTools Code Cleanup adds your customized error handling to every procedure. After running the Code Cleanup tool, the code above could look like this:

```
Function DeleteFile()
  On Error GoTo PROC_ERR

  Kill "C:\Myfile.txt"
  Exit Function

PROC_ERR:
  MsgBox "The following error occurred: " & Err.Description
  Resume Next
End Function
```

Editing the Error Handling Code

Use the options on the Error Handling page of the Standards form to customize the error handling text. These settings are shared between Code Cleanup and the New Procedure and New Property Builders:



Standards Form: Error Handling

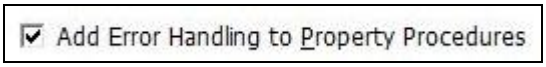
Use the options on this form's four tabs to customize the error handling.

The error handling code that you specify is inserted into your code exactly as it appears—your indentation settings are not applied to this code. Be sure to incorporate proper indentation.

Cleanup Options Tab

The Cleanup Options determine what the Code Cleanup tool does when its "Error Handling" option is checked.

Add Error Handling to Property Procedures



By default, when Error Handling is selected, it is added to all procedures (Functions and Subs). You have the option to also add error handling to property procedures (Get, Let, and Set). By default, this is selected, but some developers do not like error handling code in property procedures since they usually contain simple variable assignments. If your property procedures are more complicated and contain programming logic, they should have error handling just like other procedures.

Update Existing Tagged Total Visual CodeTools Error Handling Code

Error handling code is divided into two parts: the Error Enabler at the beginning of the procedure and the Error Handler at the end of the procedure.

Total Visual CodeTools lets you add comment tags before and after these sections to define where they begin and end. If you have these tags, Code Cleanup can update your error handling.

When Code Cleanup sees error handling in a procedure (the On Error statement), it does not apply error handling, but if it sees the error handling tags and this option is turned on, it replaces everything between the tags with the new error handling routines.

Automated Error Handling Updates

Total Visual CodeTools allows you to add comment tags around your code so that it can be automatically updated in the future:

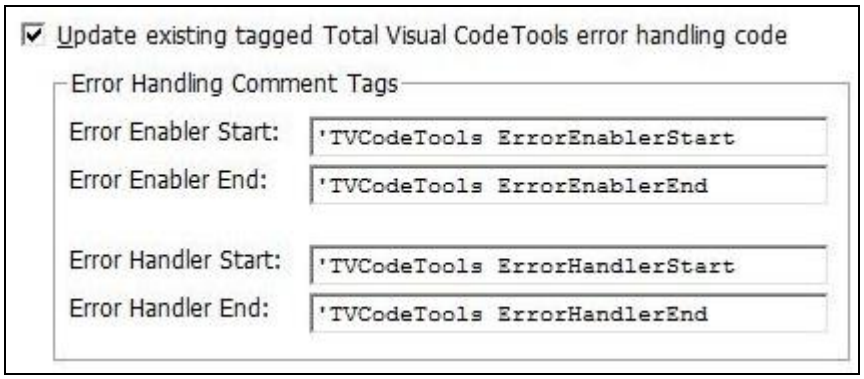
```
'TVCodeTools ErrorEnablerStart
<< Error Enabler Code >>
'TVCodeTools ErrorEnablerEnd
```

Error Enabler Start and End Comment Tags

```
'TVCodeTools ErrorHandlerStart
<< Error Handler Code >>
'TVCodeTools ErrorHandlerEnd
```

Error Handler Start and End Comment Tags

If you have these comments designating where error handling begins and ends, and decide to change your error handling code, the Code Cleanup feature can apply your changes to every procedure that has these tags. This lets you change your error enabler and handler over time from one place.



Update Option with Error Enabler and Error Handler Comment Tags

If you check the "Update existing tagged Total Visual CodeTools error handling code" option, the code between these tags is replaced.

You can customize these comment tags under the Error Handling Comment Tags section. Note that the comment tags can be indented to any level, but they must be on their own lines.

If you modify these, you need to manually adjust your error handling text to include these in the other tabs. Total Visual CodeTools does not automatically adjust those values.

If you are not interested in this feature, you can uncheck this Update option or delete the four comment tags from the error handling templates under the Module and Class tags.

Error Handling Text

Use the other three tabs to specify the error handling you'd like to use for different module types. You can use the same structure or specify different ways to handle errors in each environment. These options are used when you run Code Cleanup and the New Procedure Builder. The Property Builder only uses the Class Modules settings.

Error Handling Tokens

In addition to regular text, Total Visual CodeTools includes a feature to let you insert runtime values called **Tokens** into your error handler with the #*Letter*# syntax. Click on the Tokens drop down list to see the available options. For error handling, these tokens are most relevant:

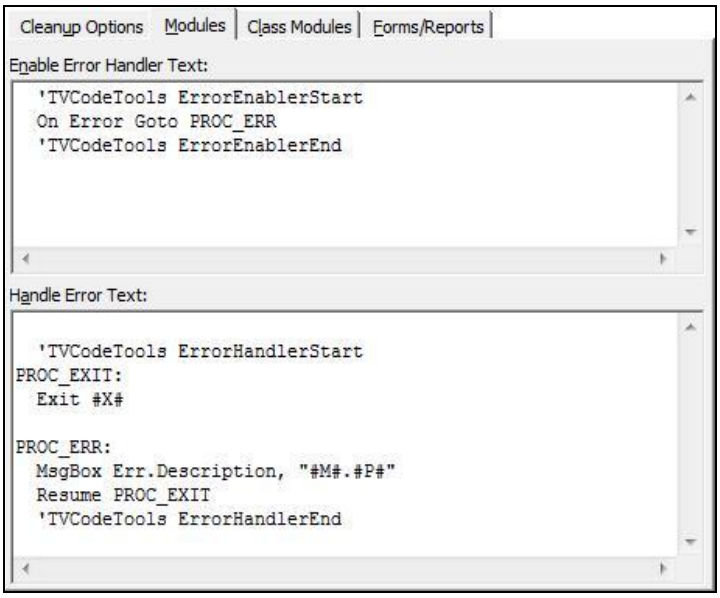
Token	Meaning
#P#	Procedure name
#X#	The word "Sub", "Function", or "Property" based on the procedure type. For example, Exit #X# becomes "Exit Sub", "Exit Function" and "Exit Property".
#M#	Module name

Error Handling Tabs

Three tabs are available:

- Modules for standards modules
- Class Modules for classes which you many want to pass the class name or raise an error. Used in the Property Builder.
- Forms/Reports which you can use the Me.Name command to give the current object name.

All three tabs are identical in appearance:



Error Handling Standards: Modules Tab

Enable Error Handler Text

The "Enable Error Handler" text box contains the code that is placed at the beginning of your procedure. This is typically the statement or statements that enable error handling in the procedure and consists of an `On Error GoTo...` statement. The default code is:

```
On Error GoTo PROC_ERR
```

You may want to designate a global variable, which controls whether error handling is turned on or off. This is particularly useful during debugging, and may look similar to this:

```
If Not gcfDebugMode Then On Error GoTo PROC_ERR
```

Where gcfDebugMode is a Public (global) constant. Set the constant to TRUE during debugging and FALSE when you ship your project. Of course, you have to manage the name and value of the constant.

If you have a procedure to track the call stack, you can pass it the name of the procedure with the #P# token.

Handle Error Text

The "Error Handler" text box holds the text of the section of code that implements the error handler. This is typically composed of a label that identifies the start of the section, and code that is executed when the error

occurs. The default error handler provides a simple way to provide the user with a message when an error occurs:

```
PROC_EXIT:
  Exit #X#

PROC_ERR:
  MsgBox "This error occurred: " & Err.Description, , _
         "Procedure: #P#"
  Resume Next
```

In this example, a message box appears and gives the user a description of the error and the procedure name. You can customize this to call your own error handling routine, or use it in a variety of other ways to process errors. By default, the code issues a Resume Next, but you may want to gracefully close your project if an unexpected error occurs.

Notice the Exit #X# statement in the text. The #X# token is automatically replaced by “Sub” or “Function” based on the procedure type. There are many tokens available for your error handler to be as flexible as possible when inserted into your procedures.

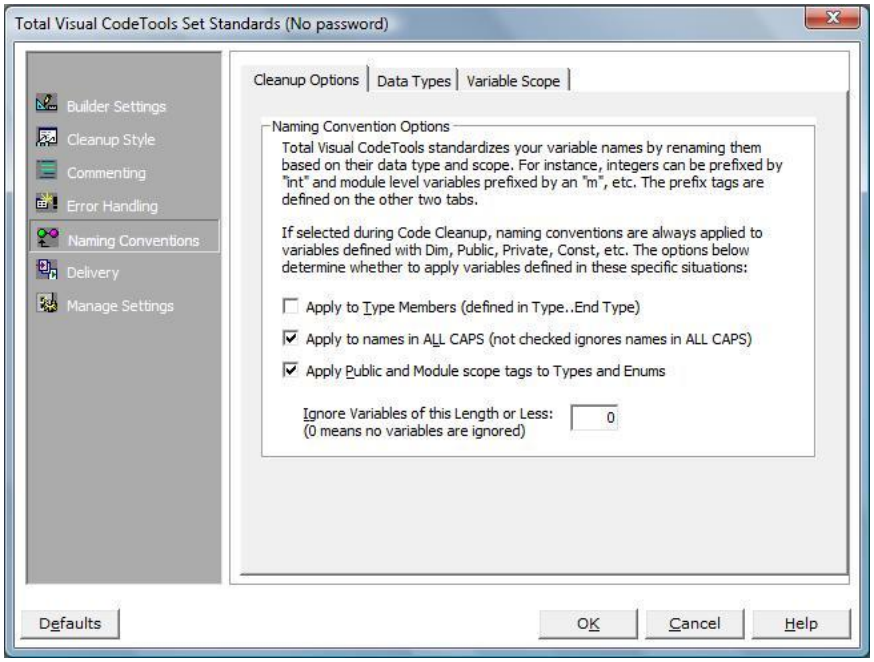
Here is an example of how the default error handling text looks in your procedure. These settings are used for both Code Cleanup and the New Procedure Builder.

```
Private Function NewProc() As Boolean
  'TVCodeTools ErrorEnablerStart
  On Error Goto PROC_ERR
  'TVCodeTools ErrorEnablerEnd


  'TVCodeTools ErrorHandlerStart
PROC_EXIT:
  Exit Function

PROC_ERR:
  MsgBox "This error occurred: " & Err.Description, , _
         "Procedure: NewProc"
  Resume PROC_EXIT
  'TVCodeTools ErrorHandlerEnd
End Function
```

Naming Conventions

The Naming Conventions section lets you specify the variable naming conventions that are used by these tools:

- Code Cleanup when “Variable Naming Conventions” is selected and Total Visual CodeTools renames variables that don’t follow your naming convention settings. Great for quickly cleaning up existing code for your whole project.
- Property Builder for the variable name defined at the General Declarations level.
- Recordset Builder for its many ADO and DAO variables



Naming Conventions Standards

Variable naming conventions make your code easier to read and maintain by adding specific characters to the beginning of variable names to designate their type. For example, instead of:

```
Dim MyName As String
```

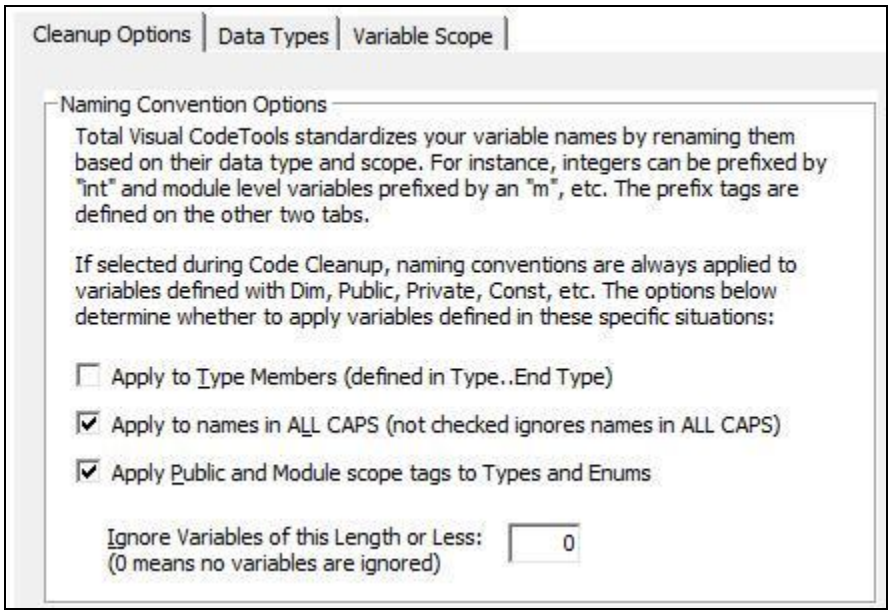
Use:

```
Dim strMyName As String
```

When you see the variable later in code, you’ll immediately know it contains a string value. Adding a naming convention not only makes it easier to identify variables, it also minimizes the chance your variables conflict with reserved words in current and future versions of VB/VBA.

Code Cleanup Options

The first tab (Cleanup Options) lets you define the scope of what names are modified by the Code Cleanup feature:

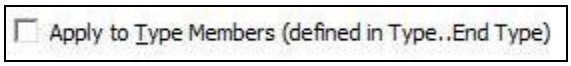


Naming Conventions Standards, Cleanup Options Tab

When the "Variable Naming Convention" option is selected for Code Cleanup, all variables are candidates for renaming. This includes variables defined by Dim, Public, Private, Const, Static, etc. You have the option to control whether this applies to other situations:

Type Members

A type can contain elements of different data types. If you check this option, those type elements are renamed to conform to your naming conventions. If not, they are not modified.



For instance, this

```
Type CustomerInfo
  CustomerID As Long
  Name As String
End Type
```

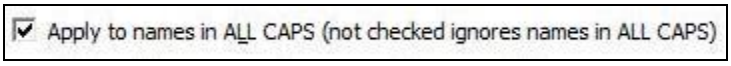
Becomes

```
Type CustomerInfo
  lngCustomerID As Long
  strName As String
End Type
```

and all references to those type elements are updated.

ALL CAPS

This option is selected by default, so that all variables are renamed regardless of whether they are in ALL CAPS.



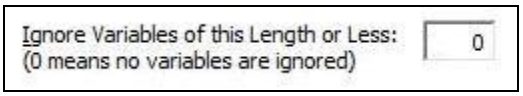
However, there are situations where you may not want to rename variables. In particular, variables used in Windows API calls. If you would like to prevent variables that are in ALL CAPS from being renamed, uncheck this option.

Public and Module Scope Tags to Types and Enums

Types and Enums generally don't have scope tags on them (i.e., "g" and "m") but you can optionally add them. If you select this option, the "g" tag is applied if the type or enum has public scope, and "m" if it scopes to the module (i.e., it is explicitly defined as Private).

Ignore Variables of this Length or Less

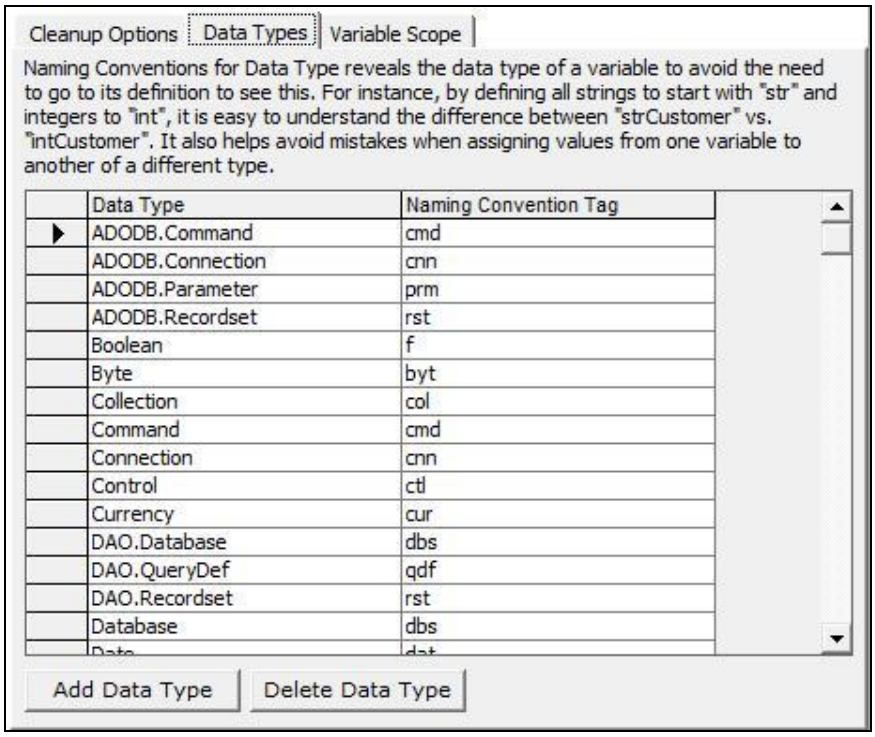
Some developers use small variable names like x, i, j, k, etc. for counters or temporary values and do not want them renamed.



This option lets you exclude variables from the naming cleanup by telling the Code Cleanup feature to ignore all variables of this length or less. A value of 0 will rename all variables to meet the naming convention.

Data Types

The Data Types tab lets you specify the naming convention tag for variables defined for each data type. By default, Total Visual CodeTools uses the naming conventions we use at FMS, which are generally accepted in the VB/VBA community, but you can customize these to match your naming convention:



Naming Conventions Standards, Data Types Tab

For example, if you want all integer variables to be tagged with the text **int**, type **int** into the Naming Convention Tag column next to the Integer data type. If you do not want to apply a naming convention for a particular type, blank the Naming Convention Tag value.

When the tag is added to your variable name during Code Cleanup, the first letter of the variable is capitalized. For example:

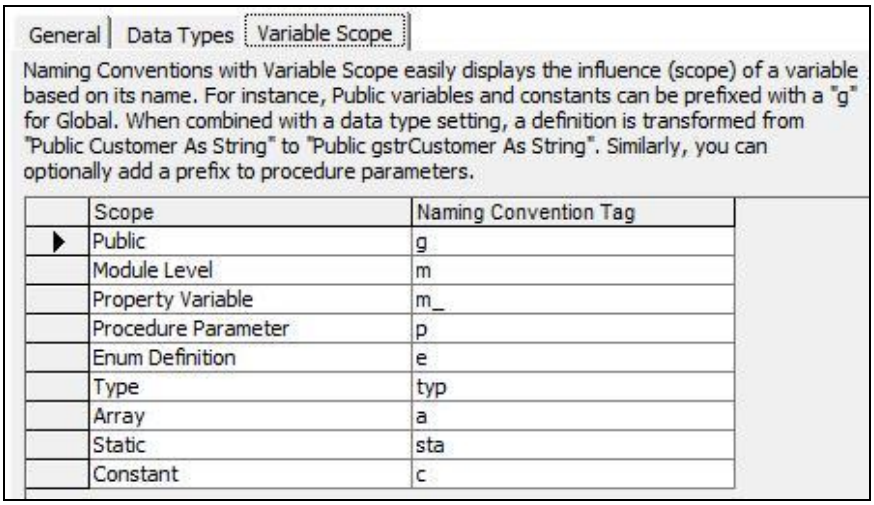
```
ORIGINAL:
Dim name As String, Hired As Date, salary As Double

CLEANED:
Dim strName As String, datHired As Date, dblSalary As
Double
```

Every use of the variable is renamed. If a variable already starts with the correct tag, it is not renamed.

Total Visual CodeTools comes with a set of standard data types, and ADODB and DAO objects. If you'd like to add your own data type, press the [Add Data Type] button to add it.

Variable Scope

The Variable Scope tab lets you customize the tags for variable scope and non-variable settings:



Naming Conventions Standards, Variable Scope Tab

Public

The Public tag is for variables declared as public or global which are “visible” to all of your modules. By default the value is “g” for global.

```
ORIGINAL:
Public Name As String

CLEANED:
Public gstrName As String
```

With a naming convention for public variables, you prevent the same variable name being defined at different levels (e.g. public, module, and procedure). It also makes it very easy to identify the scope of these variables when you see them in your code. Understanding that the variable comes from beyond the current procedure makes it clear that modifications to it may impact other procedures.

Module Level

The Module Level tag is used on variables that are defined with a Dim or Private statement in the general declarations section of a module. By default the value is “m” for module:

```
ORIGINAL:
Private strName As String
Dim lngID As Long

CLEANED:
Private mstrName As String
Dim mlngID As Long
```

Similar to the naming convention for public variables, module level tags make it easy to identify variables defined for the current module and may be used by multiple procedures in the module.

Property Procedure Class Variables

A naming convention that is gaining acceptance is the use of "m_" to prefix general declaration level variables in class modules that hold the value of the class properties. The purpose of this prefix is to create a clear distinction between regular module level variables used by the class procedures versus variables holding values that may be referenced externally through a property procedure.

By default the value is "m_":

```
ORIGINAL:
Private mstrCaption As String
Private lngColor As Long

CLEANED:
Private m_strCaption As String
Private m_lngColor As Long
```

During Code Cleanup, Total Visual CodeTools examines the property Get/Let/Set statements. If it sees that the same variable is assigned or referenced for the corresponding property statements, it identifies it as a Property Procedure Class Variable.

If your property statements are more complex and not the simple assignment of a value to a variable, Total Visual CodeTools may be unable to determine the variable you use to store the property value. In such cases, it does not apply this naming convention, and the module tag is applied.

Notice how mstrCaption is converted to m_strCaption and not m_mstrCaption. This is because if a tag ends with a "_", a check is first made to see if the other characters match. If so, just the "_" is inserted.

Enum Definition

You can also apply naming conventions to Enums—the default is "e":

```
ORIGINAL:
Enum SpecialList
  first
  second
End Enum

CLEANED:
Enum eSpecialList
  first
  second
End Enum
```

This only impacts the enum definition, not its elements.

Type Definition

User defined types can also have a naming convention—the default is "typ":

```
ORIGINAL:
Type Person
  name As String
  birthday As Date
End Type

CLEANED:
Type typPerson
  name As String
  birthday As Date
End Type
```

This only impacts the type definition, not its elements.

Array

Arrays can also have a naming convention in addition to a naming convention applied to the data within the array. The default is "a":

```
ORIGINAL:
Dim Names() As String

CLEANED:
Dim astrNames() As String
```

The scope of the array may also add another tag. For instance, if the example were a public array, it would be named gastrNames().

Constant

Constants can also have a naming convention in addition to a naming convention applied to the data type if it is explicitly defined.

The default is "c":

```
ORIGINAL:
Const InvoiceTable = "Invoices"
Const VersionDate As Date = #11/8/08#

CLEANED:
Const cInvoiceTable = "Invoices"
Const cdtVersionDate As Date = #11/8/08#
```

The scope of the array may also add another tag. If the examples were public constants, they would start with a "g."

Procedure Parameter

In addition to public and module level tags, some people like to tag the procedure parameter variables. This is particularly helpful in complex procedures where there may be confusion among parameters (arguments) passed to the procedure versus the variables defined in the procedure.

By default, the tag is "p":

```
ORIGINAL:
Sub MyCode(lngID As Long, strName As String)

CLEANED:
Sub MyCode(plngID As Long, pstrName As String)
```

Of course all references to these variables within the procedure are also renamed.

The use of this convention is particularly helpful for making sure you don't accidentally change the value of a procedure parameter. By default, parameters are passed by reference (ByRef), which means that changes to their values within the procedure also affect the calling procedure. This may or may not be what you want, so it's good to know when you are doing it.

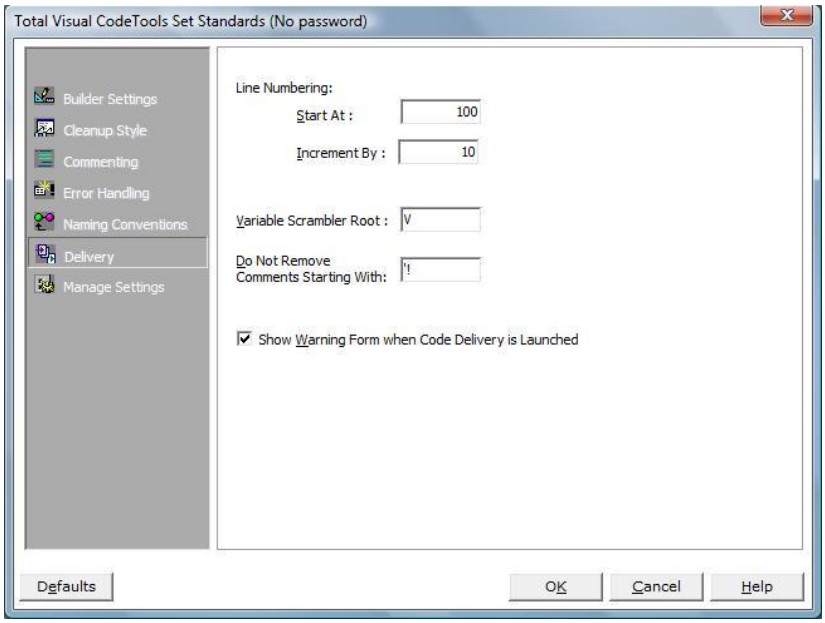
Static

Static variables, which can only be defined within a procedure, can also have a naming convention. The default is "sta":

```
ORIGINAL:
Static lngCounter As Long

CLEANED:
Static stlngCounter As Long
```

Delivery

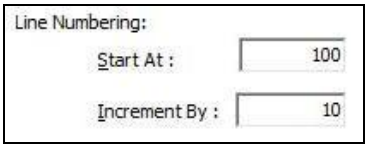
The Delivery section lets you specify the options for the Code Delivery feature:



Standards Form: Delivery

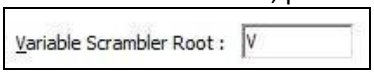
Line Numbering

If the "Line Numbering" option is selected from Code Delivery, a number is assigned to each line of your project. The numbering starts for each module or class with the Start At value, with subsequent numbers incremented by the amount specified:



Variable Scrambler Root

When you choose the "Variable Scrambling" option from Code Delivery form, the variables are renamed to this value, plus a number.

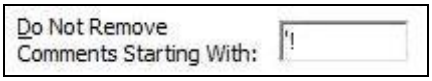


By default, the value is "V" which renames the project's variables to "V1", "V2", "V3", etc.

Do Not Remove Comments Starting With

You can optionally choose to prevent the "Remove Comments" option from removing all comments. For instance, you may have ownership or copyright information at the top of each module that you need to preserve.

Specify the characters that indicate the comments to preserve:

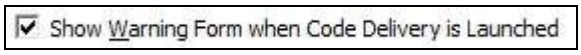


By default, the tag is `'!` (single quote and exclamation point), and comments that start with these characters are not removed. This lets you preserve things like:

```
'! Copyright (c) FMS, Inc.  All Rights Reserved.
```

Show Warning

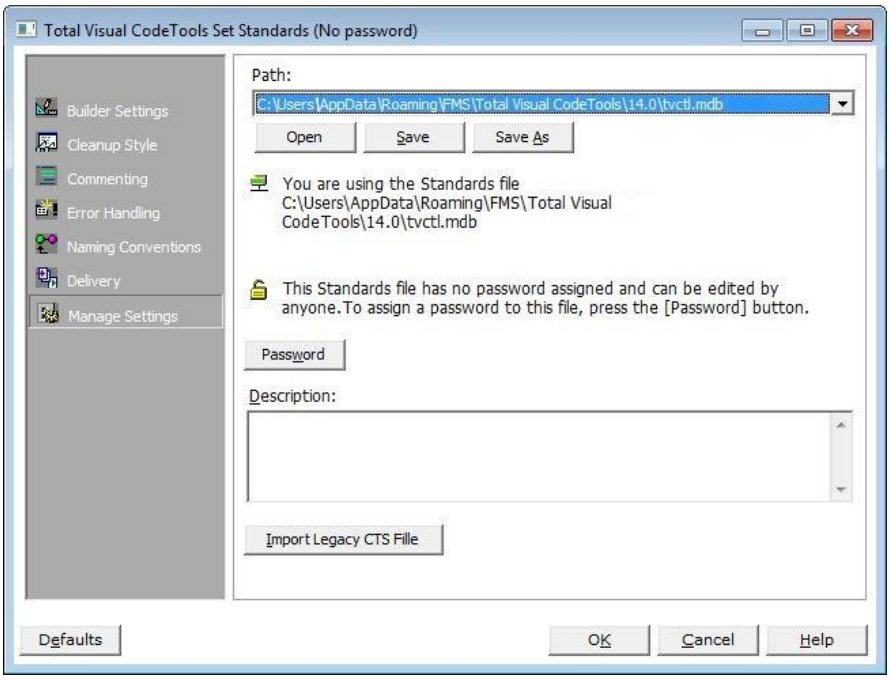
When the Code Delivery feature is started, a warning message appears to confirm you've backed up your project. This option lets you hide that warning. If you hide it, you can turn it back on by checking this box:



Managing Settings Files

This section covers the management of Shared Standards files, adding password-protection, making backups, restoring defaults, and importing settings from earlier versions of Total Visual CodeTools.

Select the Manage Settings option from the Standards form:

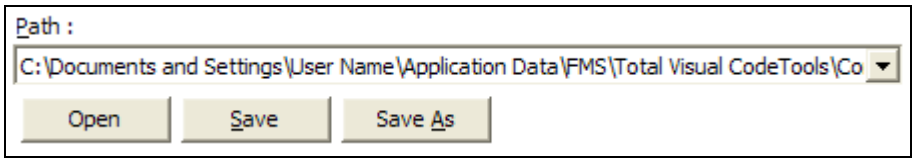


Standards Form: Manage Settings

Shared settings are stored in a standards file, located in the path specified. For an overview of what is stored in this file, read the section **Standards Architecture** on page 24.

Specifying the File Name

Specify the file where your settings are stored:



Path and Name of the Settings File

By default, the file is located in your Application Data folder. To use another file, like a shared file for the team, press [Open] and select that file. For situations where a shared settings file is available for all developers, each developer must point to that file here.

The [Save] button saves your current settings (including any changes you've made since invoking Standards) to the file.

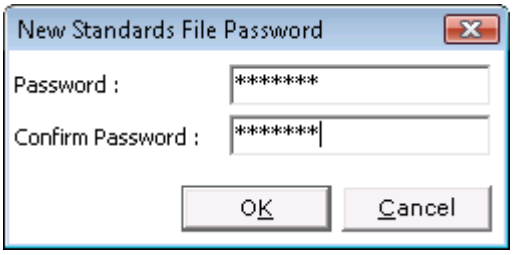
The [Save As] button saves the current settings under a new file name which becomes the current file. Add a description in the description box to identify the version you created. This file is not password protected.

Password-Protecting Shared Standards Files

To prevent unwanted changes to your standards, you can assign a password to your Shared Standard file. By assigning a password, you restrict people from modifying the Standards settings. This does not affect these Standards which are user specific: user name and initials.

The password is more of a convenience rather than a real security feature. Even if you assign a password to a Shared Settings file, there is nothing to prevent a developer from pointing to another (non-password-protected) settings file. The password-protection is designed to prevent developers from making unintentional changes to the team or enterprise standards.

To implement password protection, press the [Password] button:



Password Assignment

Type in and confirm your password, then press [OK].

If you load the Standards form based on a password protected file, you'll need to press the [Password] button to enter the password before being able to edit the settings.

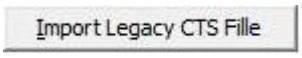
If you place your Shared Standards file on a shared drive, add a password to avoid other Total Visual CodeTools users from inadvertently changing your settings.

Clearing Passwords

Use the Save As feature to clear your password. The new file is not password protected.

Import Legacy CTS File

In prior versions of Total Visual CodeTools, standards were saved in CTS text files. You can import those settings and overwrite your existing settings from them.



You can review the new values before pressing OK to save them.

Making Backups of Shared Standards Files

Because Shared Standards file contains a lot of information, recreating it manually would be a time-consuming and error prone process. To prevent this from happening, make a backup of your Shared Standards file. To do this, simply copy the tvctl.mdb file to your backup media.

Restoring Defaults

At any time, you can restore the current Shared Settings to their default values by pressing the [Defaults] button at the bottom of the Standards form. Because this action overwrites any current values, you should use the [Save As] button to save your current settings before restoring defaults.

Shared Settings Scenarios

The Total Visual CodeTools Shared Standards architecture is designed to be both easy to use and flexible. This section contains the information you need to get up and running.

If you're simply using Total Visual CodeTools on your own on one project, the one settings file is sufficient. However, there are other scenarios where our Settings file architecture supports:

Single Developer, Multiple Projects

If you are working on multiple projects, you can use multiple Shared Standards files to accommodate the individual requirements of each project. For example, each project may require a particular type of commenting and error handling format.

Use the Manage Settings page on the Standards form to select the Shared Standards you previously saved.

Multiple Developers, Connected to the Same Network

In this scenario, multiple developers are connected to the same network or shared drive. Each developer has an installation of Total Visual CodeTools and you want to configure the Shared Standards so every developer uses the same settings.

To accomplish this, you must first create the Shared Standards file, then have each user of Total Visual CodeTools to point to that file.

Note that Total Visual CodeTools does not attempt any locking on the Shared Settings file. If two developers are editing the file simultaneously, the last developer to save wins.

Refreshing Shared Settings

The design of Total Visual CodeTools presumes shared settings are not changed very often and are modified in a very deliberate and controlled manner. Rather than checking the file ever time a builder is invoked, Total Visual CodeTools only loads the file when it starts.

Shared settings are not automatically refreshed by each user as changes are made. If you are connected to a shared settings file and another Total Visual CodeTools user changes the file, your copy of Total Visual CodeTools does not automatically see the changes. To see the most current settings, reload Total Visual CodeTools or press the [Open] button on the Manage Settings page, and re-select your settings file.

Settings Cross-Reference

This section lists each value stored under Standards by page. It lists each page, the option, what tool(s) it applies to, and whether it's a shared setting. Shared settings are stored in a file that can be password protected and used by multiple developers. Non-shared settings are local to the user and stored in their registry.

Standards Page	Option	Applies To	Shared
Builder Settings	Tab Width	All Code Builders Cleanup	Yes
Builder Settings	Max Line Width for Builder Generated code	Long Text Builder Message Box Builder Recordset Builder	Yes

Standards Page	Option	Applies To	Shared
Builder Settings	Menu Location	All	No
Cleanup Style	Standardize Indenting/Split Colon separated statements Split single line If statements Split single line declarations Lines between procedures Maximum consecutive blank lines	Cleanup	Yes
Cleanup Style	Procedure sorting enabled Procedure sort order Property sort order Show warning form	Cleanup	Yes
Commenting, General Tab	User Name for Comments User Initials	Used by tokens in comments and error handling	No
Commenting, General Tab	Optional Comment to Add in the Block Commenter	Block Commenter	Yes
Commenting, General Tab	Place Procedure Comments	New Procedure Builder New Property Builder Cleanup	Yes
Commenting, General Tab	Format for inserted dates Format for inserted time	Used by tokens in comments and error handling	Yes
Commenting, Cleanup Mod Tab	Always Add Comments to Module even if Comments exist	Cleanup	Yes
Commenting, Cleanup Mod Tab	Module Comment Template	Cleanup	Yes
Commenting, Cleanup Proc Tab	Always Add Comments to Function or Sub even if comments exist	Cleanup	Yes
Commenting, Cleanup Proc Tab	Template for Functions Template for Subs	Cleanup	Yes
Commenting, Cleanup Prop Tab	Always Add Comments to Property even if they exist	Cleanup	Yes
Commenting, Cleanup Prop Tab	Template for Property Get Procedures Template for Property Let/Set Procedures	Cleanup	Yes

Standards Page	Option	Applies To	Shared
Commenting, New Proc Tab	Template for Builder Functions Template for Builder Subs	New Procedure Builder	Yes
Commenting, New Prop Tab	Template for Property Get Procedures Template for Property Let/Set Procs	New Property Procedure	Yes
Error Handling, Cleanup Options Tab	Add Error Handling to Property Procs Update existing Total Visual CodeTools error handling code Error Handling Comment Tags	Cleanup	Yes
Error Handling, Modules Tab	Enable Error Handler Text Handler Error Text	Cleanup New Procedure Builder	Yes
Error Handling, Class Modules Tab	Enable Error Handler Text Handler Error Text	Cleanup New Procedure Builder New Property Builder	Yes
Error Handling, Forms/Reports Tab	Enable Error Handler Text Handler Error Text	Cleanup New Procedure Builder	Yes
Naming Conventions, Cleanup Options Tab	Apply to type members Apply to names in ALL CAPS Apply public and module scope tags to Types and Enums Ignore variables of the length or less	Cleanup	Yes
Naming Conventions, Data Types Tab	Data type tags	New Property Builder Recordset Builder Cleanup	Yes
Naming Conventions, Variable Scope Tab	Variable Scope tags	New Property Builder Cleanup	Yes
Delivery	Line Numbering Start At Line Numbering Increment by Variable Scrambler Root Leave comments enabled which are marked with Show warning form	Delivery	Yes
Manage Settings	Path	Standards	No
Manage Settings	Description	Standards	Yes

Chapter 4: Code Builders

Developing VB/VBA applications involves many common operations that are tedious and error-prone. Total Visual CodeTools includes several Code Builders to help you accomplish common tasks at the touch of a button. By automating these tasks, you can cut the time you spend on tedious processes, and focus on the bigger picture. This chapter provides details about each of the Code Builders.

Topics in this Chapter

Builders Overview

Total Visual CodeTools offers several Code Builders that simplify the development process. The following Builders are available:

New Procedure Builder

The New Procedure Builder allows you to quickly create new procedures that adhere to your set of coding standards. It uses the comment headers and error handling that you specify, so you don't have to do it manually every time.

New Property Builder

The New Property Builder makes it simple to build property Let, Get, and Set procedures in your class modules. The properties use your commenting and error handling standards, and creates a module level variable.

Long Text/SQL Builder

Converting a block of text or long SQL string into a module is time consuming if you want to create multi-line statements with line continuations and handle quotes. The Long Text/SQL Builder automates the process with special handling of SQL strings to break lines at SQL terms such as FROM, WHERE, GROUP BY, etc.

Recordset Builder

When writing code for a database, one constantly creates recordsets against a table, query, or stored procedure, but this process is cumbersome and error-prone. The Recordset Builder generates ADO or DAO code for the object and fields you select with options to browse, add or edit the recordset.

Message Box Builder

Message boxes are a common, but there is no way to create them visually, or to test them without running your code. Even experienced developers need to refer to the manual or help system to determine the values of icons and button options. The Message Box Builder lets you create a message box visually, and generates the code for you including code to handle the user's choice.

Select Case Builder

Typing Select...Case statements can be a tedious process, especially when there are many conditions. The Select Case Builder allows you to create Select...Case statements quickly and assign a variable for each case.

Copy Control Code Builder

The Copy Control Code Builder copies a control's code to another control. This helps you increase your productivity and accuracy, and reduces the frustration of repetitive code writing.

Format Builder

The VB/VBA Format function allows you to customize the way values are displayed and printed, but it may be difficult toremember the arguments and syntax for this function. The Format Builder makes it simple to format expressions according to common or custom date, number, string, and currency formats.

DateDiff Builder

The VB/VBA DateDiff function calculates the time intervals between two specified dates, but it is difficult to remember the syntax and arguments of the function. The DateDiff Builder builds code to calculate date and time differences, including years, quarters, months, weeks, days, weekdays, hours, minutes, seconds, and more.

Using Generated Code

After using the Builders to generate code, you can insert it directly in your current module, copy the results to the clipboard, send the results to a file, or send the results to an external editor. The options are at the bottom of each builder:

Send To Options

The external editor (NotePad in the example above) is set in the Standards form, under Builder Settings. For more information, see **External Editor** on page 28.

Project

Total Visual CodeTools pastes the generated code in your project at the position of your cursor in the current module.

To use the “Project” option, you must have your cursor in the module window prior to opening a Code Builder. The “Project” option is disabled if there are no modules open in the workspace, or if the current module is read-only.

Clipboard

The “Clipboard” option is useful if you start the Builder when you are not editing a module at the exact location where you want to add the new code. After sending the results to the clipboard, you can open a module, move to the appropriate location, and paste the results (using the Edit|Paste menu item or pressing [Ctrl]+[V]).

File

In some cases, you may want to save the generated code to a file for later use. When you choose the “File” option, a dialog prompts you to choose the destination of the file. Press [OK] to create the file.

External Editor

Choose the “External Editor” option to send the generated code to an external editor (such as Notepad, WordPad, or Word) to save or edit it. These options are set under the Standards section.

New Procedure Builder

The New Procedure Builder allows you to quickly create procedures with your commenting and error handling structures. You specify the procedure name, type, scope, and return type, and the Builder generates the code according to your standards.

Using the New Procedure Builder, you can generate the following procedure simply by typing the procedure name “MyRoutine”—the Builder takes care of the rest:

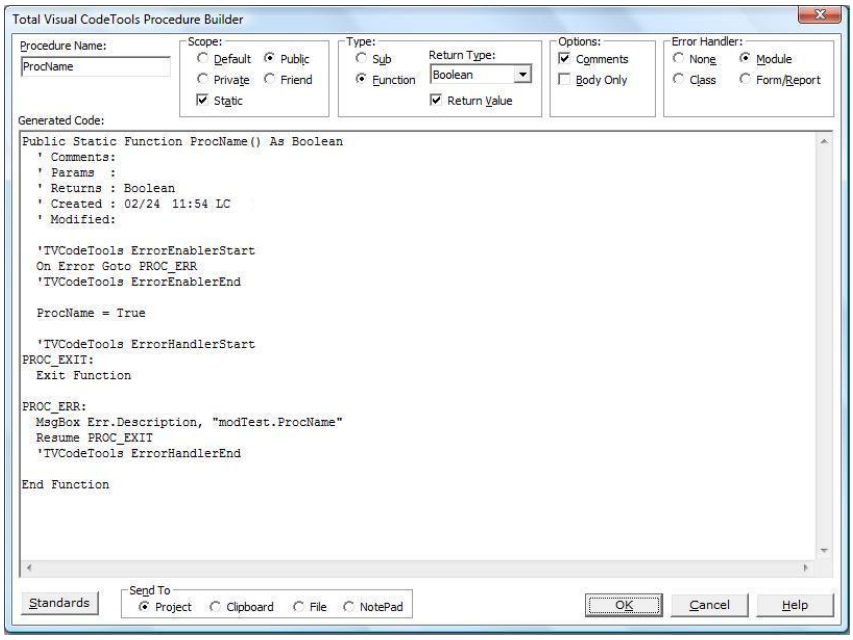
```
Function MyRoutine() As Boolean
  ' Comments:
  ' Params  :
  ' Returns : Boolean
  ' Created : 02/17 12:32 LC
  ' Modified:

  'TVCodeTools ErrorEnablerStart
  On Error Goto PROC_ERR
  'TVCodeTools ErrorEnablerEnd

  MyRoutine = True

  'TVCodeTools ErrorHandlerStart

PROC_EXIT:
  Exit Function

PROC_ERR:
  MsgBox Err.Description, "ModuleName.MyRoutine"
  Resume PROC_EXIT
  'TVCodeTools ErrorHandlerEnd

End Function
```

Your customized comments are added below the function definition, and error handling is added automatically with the error enabler lines at the top (On Error…) and the Error Handler section at the bottom (which includes a reference to the procedure name). If it's a function, you can specify the return type and variable name.

There are some "funny" comments surrounding the error enabler and error handler designating where they begin and end. These comments are optional, but if you keep them, the Code Cleanup feature can automatically update your error enabler and error handling sections in the future. See **Update Existing Tagged Total Visual CodeTools Error Handling Code** on page 47 for more information.

Invoking the Procedure Builder

To open this Builder, select "New Procedure" from the Total Visual CodeTools menu or toolbar:



New Procedure Builder Form

As you change options, the code in the "Generated Code" text box automatically updates to show your selections. The following options are available:

Procedure Name

Enter the name of your new procedure.

Procedure Scope and Static

You can specify the scope of your procedure and whether it should be static:

- Public lets code outside the module to see it.
- Private makes the procedure only visible within the module.
- Friend is an option for class modules
- Default defines the procedure without a public/private/friend statement. VB/VBA interprets this differently based on the module type. For regular modules, it is Public, for class modules (including Access forms and reports), it is Private.

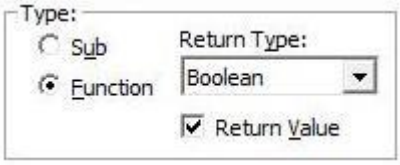
For Best Practices, it's preferable to explicitly declare the procedure as Public, Private, or Friend to avoid any confusion. See **Use Narrow Variable Scoping** on page 166.

Static

A static procedure preserves its local variables between procedure calls. Select this check box to add the Static keyword to your procedure definition. Make sure you understand the implications of making the procedure static before selecting this option— you may want to consider using static variables instead of static procedures.

Procedure Type

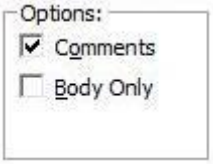
Specify whether the procedure is a "Sub" or "Function":



For functions, you can specify the data type for the return value. Checking the Return Value option adds a line in your function with the procedure name and a default value based on the Return Type.

Options

There are two additional options:



Comments

Select this to add your comment structure into the new procedure. The Comments are customized under Standards, Commenting, New Proc tab (see page 35). Through the use of Tokens, you can include your procedure name, current date/time, developer initials, etc. into your comment.

Body Only

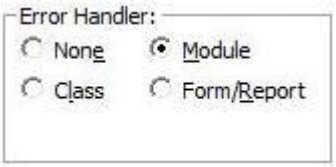
There may be cases where you only want the Procedure Builder to generate the body of the new code, and not include the procedure declaration or the

End Sub/Function line. Select this check box to generate the body of the procedure only.

This is particularly useful for event procedures where the procedure declaration is created for you by VB/VBA. For example, if you double-click on a control on a Visual Basic form, the code window opens and Visual Basic creates a blank procedure for you—the procedure parameters for that event are automatically created along with the End Sub. In such a case, you would not want to have the Total Visual CodeTools Procedure Builder overwrite the existing procedure definition, you simply want to insert the Body of the procedure with your comments and error handling..

Error Handler

Use the "Error Handler" option to specify whether you want to add error handling to your procedure and if so, which standard to use:



None omits this while the other selections correspond to the tabs on the Standards, Error Handling settings and can include the use of Tokens for your procedure name, procedure type, module name, etc. See **Error Handling Tabs** on page 48 for details.

By defining your error handling code uniquely for the three tabs, you can easily select them for the different types of procedures you need to create.

Using the Generated Code

Once the Procedure Builder generates the code you need, select your "Send To" option and press the [OK] button (see **Using Generated Code** on page 71 for more information).

New Property Builder

Property Statements expose a class module's internal data. Creating a property involves one to two separate VB/VBA procedures. The Total Visual CodeTools Property Builder makes it easy to create property statements that conform to your standards. You specify the property name, type, and

module-level variable, and the Builder generates the code according to your standards.

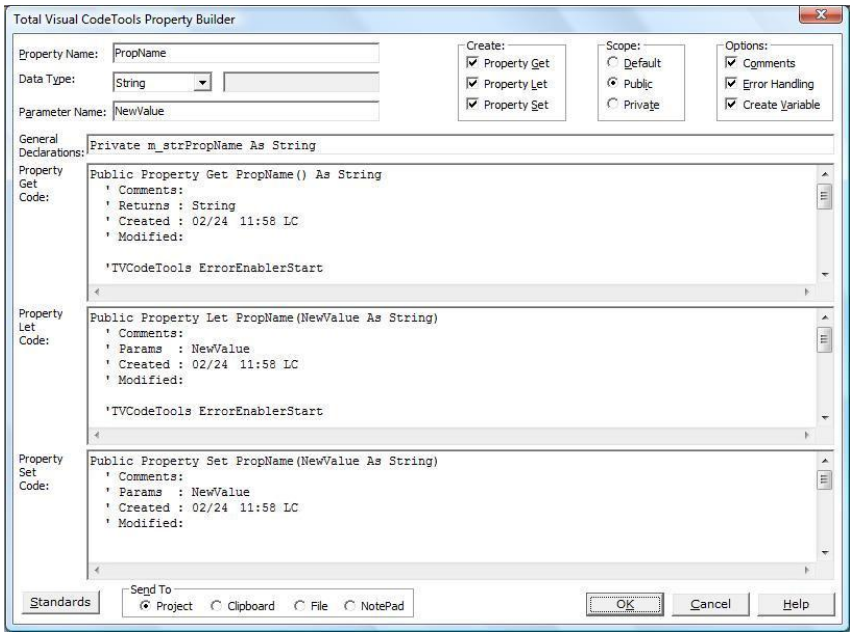
Using the New Property Builder, you can generate the following procedures simply by typing the property name “InputTable”:

```
Private m_strInputTable As String

Public Property Get InputTable() As String
  ' Comments:
  ' Params  :
  ' Returns : String
  ' Created : 10/05 16:21 LC
  ' Modified:

  'TVCodeTools ErrorEnablerStart
  On Error Goto PROC_ERR
  'TVCodeTools ErrorEnablerEnd

  InputTable = m_strInputTable

  'TVCodeTools ErrorHandlerStart

PROC_EXIT:
  Exit Property

PROC_ERR:
  Err.Raise Err.Number
  'TVCodeTools ErrorHandlerEnd
End Property

Public Property Let InputTable(NewValue As String)
  ' Comments:
  ' Params  :
  ' Created : 10/05 16:21 LC
  ' Modified:

  'TVCodeTools ErrorEnablerStart
  On Error Goto PROC_ERR
  'TVCodeTools ErrorEnablerEnd

  m_strInputTable = NewValue

  'TVCodeTools ErrorHandlerStart
PROC_EXIT:
  Exit Property

PROC_ERR:
  Err.Raise Err.Number
  'TVCodeTools ErrorHandlerEnd
End Property
```

You should understand how to write Get, Let, and Set property procedures before using this builder. Refer to your VB/VBA user guides for writing class properties for more details.

Invoking the New Property Builder

To open this Builder, select “New Property” from the Total Visual CodeTools menu or toolbar:

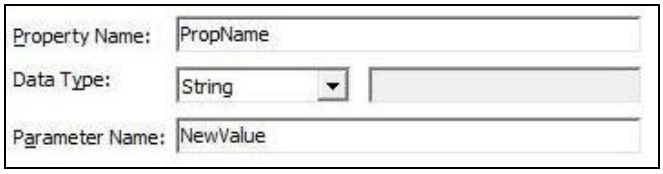


New Property Builder Form

As you change options, the code in the boxes from General Declarations and below automatically update to show your selections. The following options are available:

Initial Options

Here is the initial set of options:



Property Name

Enter the name of your new property.

Data Type

Select the data type of the property you are creating from the list. If you select <Other>, enter the name of the data type next to it.

For instance, to specify a Recordset, set the Data Type combo box to "Other" and type the word "Recordset" in the text box.

If you want to create a Set property procedure, select a data type that is valid for Set operations, such as Variant, Object, or Other.

Parameter Name

The variable name (parameter) being passed to the Let or Set Property procedure.

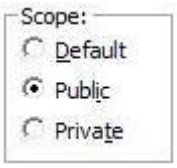
Create

Specify any combination of the Let/Set/Get property procedures to build by clicking the appropriate check box.

If you select "Property Set", make sure your data type is valid for Set operations.

Scope

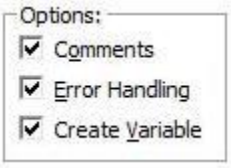
Specify the scope of the property: Default, Public, or Private.



If you want the class data to be accessible, you should generally choose Public. In classes, if Public is not specified, it's treated as Private.

Options

Three additional options are available:



Comments

Select this to add your comment structure into the property procedures. The Comments are customized under Standards, Commenting, New Prop tab (see page 35). Through the use of Tokens, you can include your property name, current date/time, developer initials, etc. into your comment.

Error Handling

Select this to add your error handling structure into the property procedures. The error handling is customized under the Class Modules tab of Standards, Error Handling. It can include the use of Tokens for your property name, module name, etc. See **Error Handling Tabs** on page 48 for details.

Create Variable

The convention for using property procedures in a class module is to create a module-level variable that always contains the value of the property. Select the "Create Variable" option to declare a module-level variable to hold the property value.

By default, this variable is prefixed with "m_" as defined under the "Property Procedure Class Variable" modifier in the Naming Convention standards.

Using the Generated Code

Once the Property Procedure Builder generates the code you need, select your "Send To" option and press [OK] (see **Using Generated Code** on page 71 for more information).

Long Text/SQL Builder

When developing applications in VB/VBA, there may be situations where you need to assign long text strings or SQL strings to variables. Splitting the variable assignment into multiple lines is tedious, and particularly troubling for SQL strings that contain quotes and syntax places you'd like to break your lines.

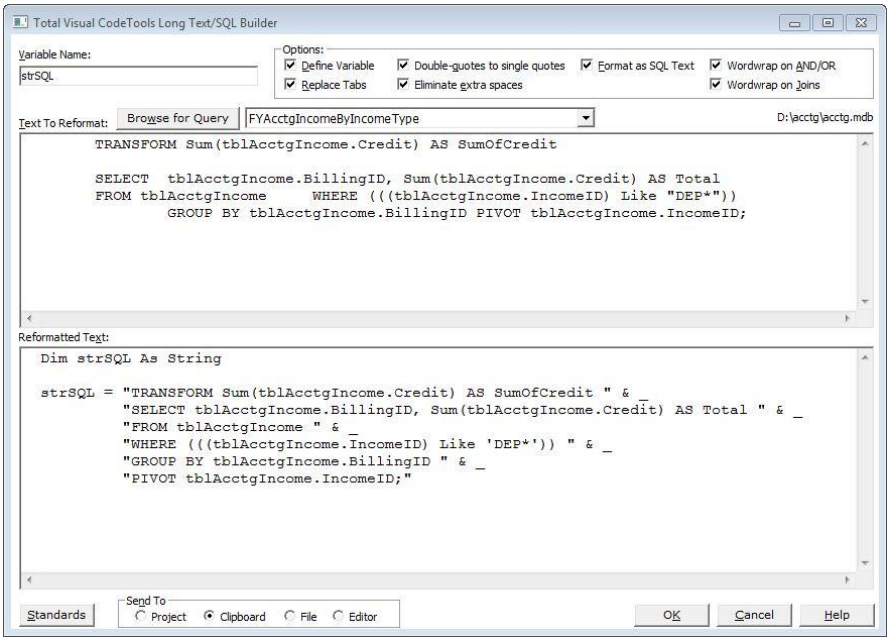
The Long Text/SQL Builder converts a block of text to code at the touch of a button. For example, this text:

```
As you change data in a database, the database file becomes
fragmented and uses more disk space than necessary.
Periodically, you can compact your database to defragment
the database file: The compacted database is usually
smaller. You can also choose to change the collating order,
the encryption, or the version of the data format while you
copy and compact the database.
```

Becomes this code:

```
strText = "As you change data in a database, the " & _
          "database file becomes fragmented and " & _
          "uses more disk space than necessary. " & _
          "Periodically, you can compact your " & _
          "database to defragment the database " & _
          "file: The compacted database is " & _
          "usually smaller. You can also choose " & _
          "to change the collating order, the " & _
          "encryption, or the version of the data " & _
          "format while you copy and compact the " & _
          "the database."
```

To open this Builder, select "Long Text/SQL Builder" from the Total Visual CodeTools menu or toolbar:

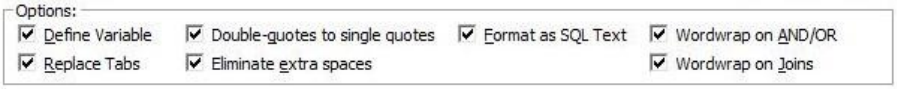


Long Text/SQL Builder Form

Enter the variable name to assign, and type or paste your string in the "Text to Reformat" box. You can also use the Browse button to retrieve the SQL string for a query in a Microsoft Access database.

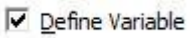
The new code is displayed in the “Reformatted Text” section.

The following options are available:



Define Variable

The “Define Variable” option adds a Dim statement for the variable name.



Double Quotes to Single Quotes

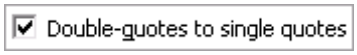
If your string contains double quotes, you cannot simply put quotes around the text and assign it to a variable. For instance, your text may be:

```
WHERE (Categories="BOOKS")
```

The builder automatically converts double quotes to two double quotes so that your string assignment works:

```
"WHERE (Categories=""BOOKS"")"
```

However, the duplicate double quotes can be hard to read.



Selecting this option converts all the double quotes to single quotes:

```
"WHERE (Categories='BOOKS')"
```

Replacing Tabs and Eliminating Extra Spaces

There are two options for eliminating tabs and extra spaces:

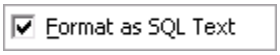
By checking “Replace Tabs”, any tabs in the text is converted to a space. This is helpful for standardizing a block of text containing a combination of spaces and tabs for indenting.

By checking “Eliminate extra spaces”, only single spaces are permitted. When more than one consecutive space is encountered, the extra spaces are deleted.

Formatting SQL Text

SQL strings stored as queries run most efficiently. However, you may need to programmatically change SQL strings and need to convert the SQL string of an existing query to code.

You could simply assign the SQL string to a variable in one line, but it is much easier to maintain and understand if it is broken into several lines and is completely visible on the screen. Dealing with quotes can also be tricky.



When you select the "Format as SQL Text" option, the Long Text/SQL Builder performs SQL syntax parsing and inserts line breaks at appropriate clauses—clauses like SELECT, FROM, WHERE, and ORDER BY always start their own lines. This makes it much easier to read and see the main components.

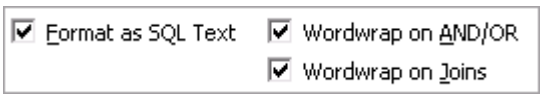
You can obtain the SQL code for a query by switching to its SQL view. Simply paste the SQL text into the "Text to Reformat" section like this:

```
SELECT DISTINCTROW Categories.[Category Name],
Products.[Product Name], Categories.Description,
Categories.Picture, Products.[Product ID],
Products.[Quantity Per Unit], Products.[Unit Price],
Products.Discontinued FROM Categories INNER JOIN Products ON
Categories.[Category ID] = Products.[Category ID] WHERE
((Categories="BOOKS")) ORDER BY Categories.[Category Name],
Products.[Product Name]
```

The SQL Builder breaks the lines and converts it to the following VBA code (the word-wrap length is set under the Standards, Builder Settings page):

```
strSQL = "SELECT DISTINCTROW Categories.[Category Name], " & _
         "Products.[Product Name], Categories." & _
         "Description, Categories.Picture, Products." & _
         "[Product ID], Products.[Quantity Per Unit], " & _
         "Products.[Unit Price], Products.Discontinued " & _
         "FROM Categories " & _
         "INNER JOIN Products ON Categories." & _
         "[Category ID] = Products.[Category ID]" & _
         "WHERE ((Categories='BOOKS')) " & _
         "ORDER BY Categories.[Category Name], Products." & _
         "[Product Name]"
```

There are a few options for SQL word-wrapping when Format as SQL Text is checked:



Wordwrap on AND/OR

If this is checked, the builder creates a new line when the AND or OR word is encountered in the SQL text. Otherwise, it is ignored and the line wraps when it reaches the maximum line width setting.

Wordwrap on Joins

If this is checked, the builder creates a new line when Joins are encountered (JOIN, INNER JOIN, or OUTER JOIN). Otherwise, the join text is ignored and the line wraps when it reaches the maximum line width.

Using the Generated Code

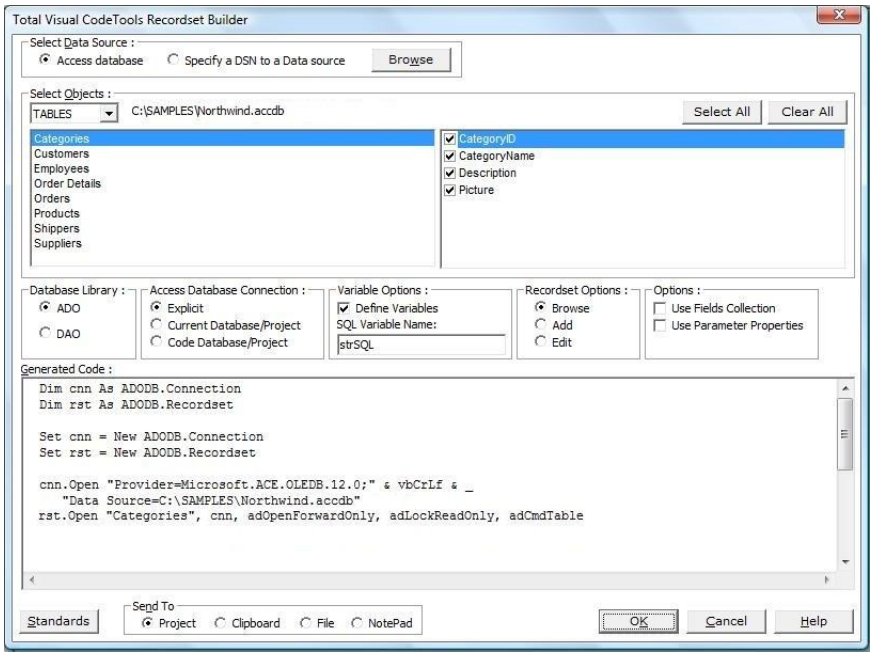
Once the Long Text/SQL Builder generates the code you need, select your "Send To" option and press [OK] (see **Using Generated Code** on page 71 for more information).

Recordset Builder

Application developers commonly need to create recordsets for tables or select queries in a database. Remembering and writing the object, field names, and ADO or DAO syntax is cumbersome.

The Recordset Builder lets you quickly generate code to open a recordset by pointing to a data source, selecting the fields, selecting whether you want to browse, add or edit the data, and use ADO or DAO syntax. It even handles parameters.

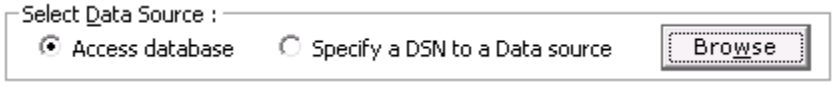
To open this Builder, select "Recordset Builder" from the Total Visual CodeTools menu or toolbar:



Recordset Builder

Select Data Source

Start by choosing where the data is. You can choose an Access database or use DSN to choose other types of data (including Access):



Access database

To generate recordset code directly against a Microsoft Access database, select this option and click [Browse] to choose an Access file. The database can be any Access database format (*.MDB, *.MDE, *.ADP, *.ADE, and *.ACCDB). To use the ACCDB format, you must have Access 2007 or Access 2010 installed on your machine.

Note that if another user or process has the database open in exclusive mode, the recordset Builder cannot open the database.

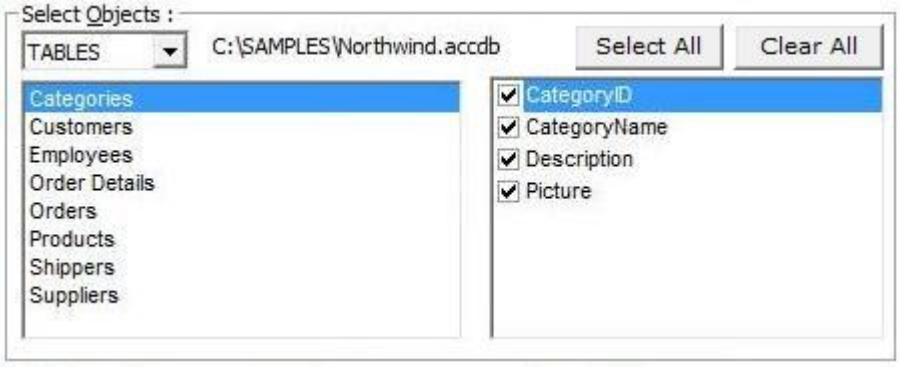
Specify a DSN to a Data Source

To connect to a database other than Microsoft Access, select Specify a DSN to Data Source and click [Browse] to choose the data source. The Recordset

Builder does not check the authenticity of the DSN that you provide and assumes you have a valid DSN with permissions to log on to the database.

Select the Object and Fields

Once the selected database is opened, the list of tables/queries and fields are displayed:

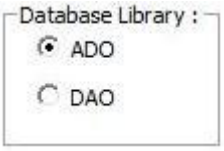


The dropdown lets you select an object type: Tables or Queries (or Views). The corresponding objects are displayed in the left column. Select an object and its fields are displayed on the right column. By default, all fields are selected. You can select a subset of fields if you want the recordset code to apply to just those fields.

There are several additional options. The code in the "Generated Code" text box automatically updates to show your selections.

Database Library

You can specify whether you want to create code using the ADO or DAO library. Your project needs to have this library included in its list of library references for the code that's generated to work.



ADO Library

The ADO option is for the Microsoft ActiveX Data Objects Library and is actually referenced by the ADODB object in your code. ADODB uses a connection object for the recordset.

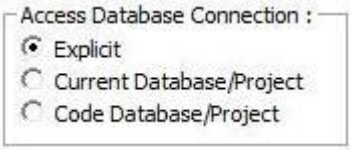
DAO Library

The DAO option is for the Microsoft DAO Object Library and is referenced by the DAO object. DAO uses a database object for the recordset.

The variable names for both ADO and DAO in the generated code are specified under Standards, Naming Conventions, Data Types tab. See page 53 for more details.

Access Database Connection

If you are using an Access database, there are some options if you are writing code that is going to be run in Access:



Explicit

Explicit is a direct reference to the database path and name. This is useful if the database is external to your application. For ADO, it creates a connection object that refers to the database. For DAO, a database object refers to the database.

Current Database/Project

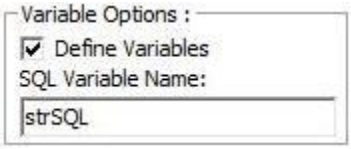
Choose this option if this code opens data in the current database. For ADO, an explicit connection object is not necessary since the recordset refers to the CurrentProject.Connection object. For DAO, the database reference is to the built-in CurrentDB object.

Code Database/Project

Choose this option if this code is for a library/add-in database to open data in itself. For ADO, an explicit connection object is not necessary since the recordset refers to the CodeProject.Connection object. For DAO, the database reference is to the built-in CodeDB object.

Variable Options

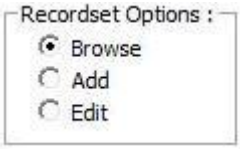
If you want a Dim statement for each variable used in the generated code, select the “Define Variables” check box.



You can also specify the name of the variable to hold the SQL string if a subset of fields is selected:

Recordset Options

You can generate recordset code that is intended for browsing, adding new records or editing.



Browse

Browse puts the recordset in a Do..Loop that goes through all the records and outputs each field value to the Immediate Window with code like this:

```
With rst
  Do While Not .EOF
    Debug.Print ![CategoryID], ![CategoryName]
    .MoveNext
  Loop
  .Close
End With
```

Add

The Add option prepares the recordset for adding a new record:

```
With rst
  .AddNew
  ![CategoryID] = 0
  ![CategoryName] = ""
  .Update
  .Close
End With
```

Example of Adding a New Record using an ADO Recordset

Each field in your datasource is listed with an assignment to zero. Once this code is in your module, change those zeros to the values you want to add to the table.

Edit

The Edit option prepares the recordset for editing the record:

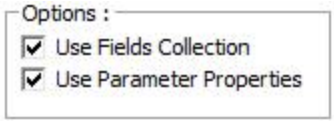
```
With rst
  .Edit
  ![CategoryID] = 0
  ![CategoryName] = ""
  .Update
  .Close
End With
```

Example of Editing an Existing Record using a DAO Recordset

Like Add, each field in your datasource is listed with an assignment to zero. Once this code is in your module, change those zeros to the values you want to replace in your record.

Other Options

There are two additional options:



Use Fields Collection

By default, the Recordset Builder refers to the field names using the ![FieldName] syntax. This is shorthand for referring to the field name in the fields collection. If you want the explicit reference, check this box and the code goes from this:

```
Debug.Print ![CategoryID]
```

to:

```
Debug.Print .Fields("CategoryID")
```

Adding the Fields collection makes each reference a bit larger, so to save space and increase readability, most developers use the ![Name] syntax.

Use Parameter Properties

If you select a query that requires parameters (to generate ad hoc results), the list of parameters is created for you to easily specify their values.

```
Set qdf = dbs.QueryDefs("[Employee Sales by Country]")
qdf.Parameters("[BeginDate]").Value = 0
qdf.Parameters("[EndDate]").Value = 0
```

Example of Parameters for a DAO QueryDef Object

If you use ADO, the parameters look like this with each parameter's details loaded in a single CreateParameter line:

```
cmd.Parameters.Append cmd.CreateParameter("[BeginDate]", _
  adDBTimeStamp, adParamInput, 19, 0)

cmd.Parameters.Append cmd.CreateParameter("[EndDate]", _
  dDBTimeStamp, adParamInput, 19, 0)
```

Example of an ADODB Command Object Appending Parameters in One Line

The Use Parameter Properties option only applies to ADO. If you select it, the details of each parameter are listed on a separate line like this:

```
  Set prm = New ADODB.Parameter
  prm.Name = "[BeginDate]"
  prm.Type = adDBTimeStamp
  prm.Size = 19
  prm.Direction = adParamInput
  prm.Value = 0          ' Insert Your Data for [BeginDate]
  cmd.Parameters.Append prm
  Set prm = Nothing
```

Example of an ADODB Command Object Assigning Parameter Properties Line by Line

The latter option takes more lines but more clearly shows each parameter property and may be easier to insert the parameter value.

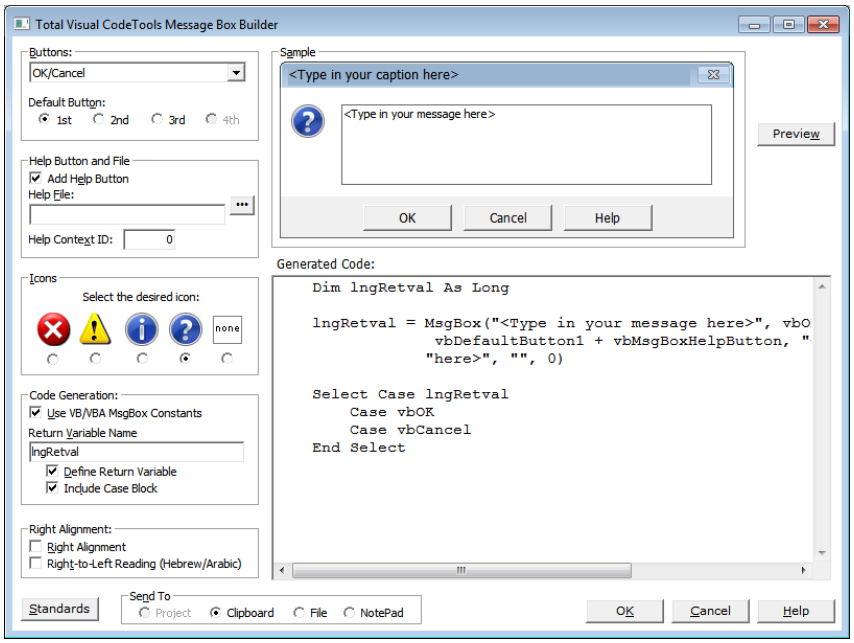
Using the Generated Code

Once the Recordset Builder generates the code you need, select your "Send To" option and press [OK] (see **Using Generated Code** on page 71 for more information).

Message Box Builder

Message boxes are simple user interface elements that are used throughout most applications with the MsgBox command. Unfortunately, even experienced developers find it difficult to remember the syntax for coding them. Rather than digging through the help file and adding up constant values, use the Message Box Builder to design them visually and automatically generate the code including code to handle the user's button response.

To open this Builder, select “Message Box Builder” from the Total Visual CodeTools menu or toolbar:



Message Box Builder Form

Sample Section

As you change options, the Sample Section displays a likeness of what your message box looks like. Use the [Preview] button to see exactly how it will appear. There is a difference because the MsgBox command uses different wordwrapping widths than the sample, fixed box.

Message Box Caption and Text

Click in the title area of the sample message box, and type the caption for your message box. Click in the message area of the sample message box, and type the text for your message box.

The “Generated Code” box shows the new code.

The following additional options are available:

Buttons

Use the combo box to select the buttons to use in the message box:

You can choose among these button combinations:

- OK
- OK/Cancel
- Abort/Retry/Ignore
- Yes/No/Cancel
- Yes/No
- Retry/Cancel

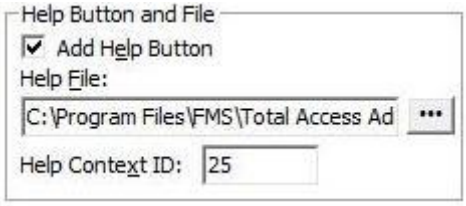
The Message Box Builder can also setup the code to process the user's button selection.

Default Button

Specify which button (from left to right) is the default.

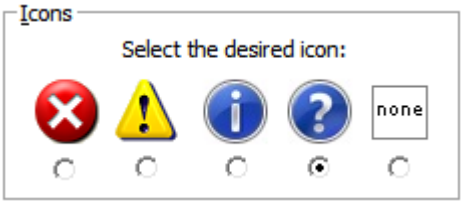
Help Button and File

You can add a [Help] button to your message box and specify the help file and context ID:



Icons

Select the icon to display on your message box:



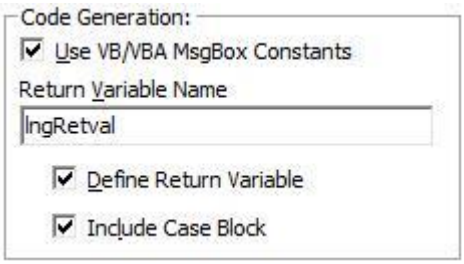
The following icons are available:

- Critical Stop
- Exclamation
- Information
- Question
- None

The sample automatically displays the icon you select.

Code Generation

There are several options to customize the code that's generated:



Use VB/VBA MsgBox Constants

The MsgBox command is passed some numbers to tell it what types of buttons to show, default values, etc. Rather than summing the options into one number and passing it, you can use the built-in VB/VBA constants to more clearly show which options are selected.

If you check "Use VB/VBA MsgBox Constants", the constants are used:

```
Call MsgBox("Message", vbOKCancel + vbCritical)
```

If you uncheck the option, a number replaces the values:

```
Call MsgBox("Message", 17)
```

If you select this option, the following built-in constants are used:

Constant Value	Constant Name
0	vbOKOnly
0	vbDefaultButton1
1	vbOKCancel
2	vbAbortRetryIgnore
3	vbYesNoCancel
4	vbYesNo
5	vbRetryCancel
16	vbCritical

32	vbQuestion
48	vbExclamation
64	vbInformation
256	vbDefaultButton2
512	vbDefaultButton3
4096	vbSystemModal

Return Variable Name

Specify a Return Variable Name for the message box to assign the user's button selection to the variable like this:

```
lngRetval = MsgBox("Message", ...)
```

Define Return Variable

If you select the "Define Return Variable" check box, a Dim statement is added above the message box to define the variable:

```
Dim lngRetval As Long
```

You may not want this if your procedure already defines the variable or if you're creating multiple message boxes in the same procedure.

Include Case Block

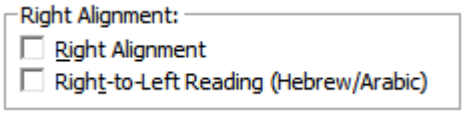
The "Include Case Block" option generates a Select Case block after the message box to handle the button pressed. For example, for a Yes/No/Cancel set of buttons, it creates this code:

```
Select Case lngRetval
  Case vbYes
  Case vbNo
  Case vbCancel
End Select
```

This feature makes it easy to add code in response to the user's selection.

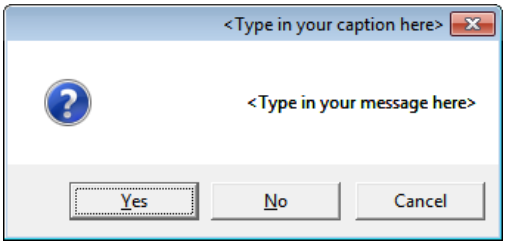
Right Alignment

Two additional options are available to right-align the message text, or to flip the orientation of the message box for right-to-left reading:



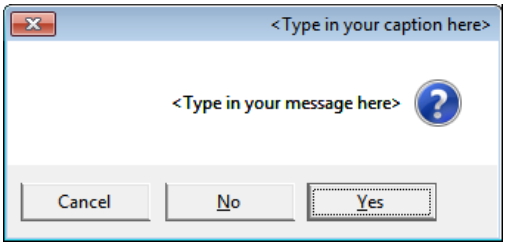
Right Alignment

The Right Alignment option right aligns the message box title in text in the window:



Right-to-Left Reading (Hebrew/Arabic)

The Right to Left Reading option orients the message box for right-to-left languages, such as Hebrew and Arabic:



Using the Generated Code

Once the Message Box Builder generates the code you need, select your "Send To" option and press [OK] (see **Using Generated Code** on page 71 for more information).

Select Case Builder

Using Select Case Instead of If

The IF..THEN..ELSE statement is nice for evaluating a variable or expression with two variables, but if it holds more than three or more values, the SELECT..CASE statement is better than a series of IF..ELSEIF..ELSEIF code:

```
If strValue = "Table" Then
  intResult = 1
ElseIf strValue = "Query" Then
  intResult = 2
ElseIf strValue = "Form" Then
  intResult = 3
End If
```

Example of Multiple IF..ELSEIF Commands

The IF..ELSEIF syntax is very flexible since you can evaluate a different expression in each ELSEIF section. But that's not good if you're evaluating the same expression because you'll need to examine each line carefully to verify that.

The SELECCT..CASE statement makes it clear that only one variable or expression is being evaluated:

```
Select Case strValue
  Case "Table": intResult = 1
  Case "Query": intResult = 2
  Case "Form":  intResult = 3
End Select
```

Example of a SELECT..CASE Statement

In the example above, the resulting variable assignment is put on the same line followed by a colon. Some people prefer that compact layout to make it easier to see the assignment, but this is a more standard result:

```
Select Case strValue
  Case "Table":
    intResult = 1
  Case "Query":
    intResult = 2
  Case "Form":
    intResult = 3
End Select
```

Example of a SELECT..CASE Statement with Results on a Separate Line

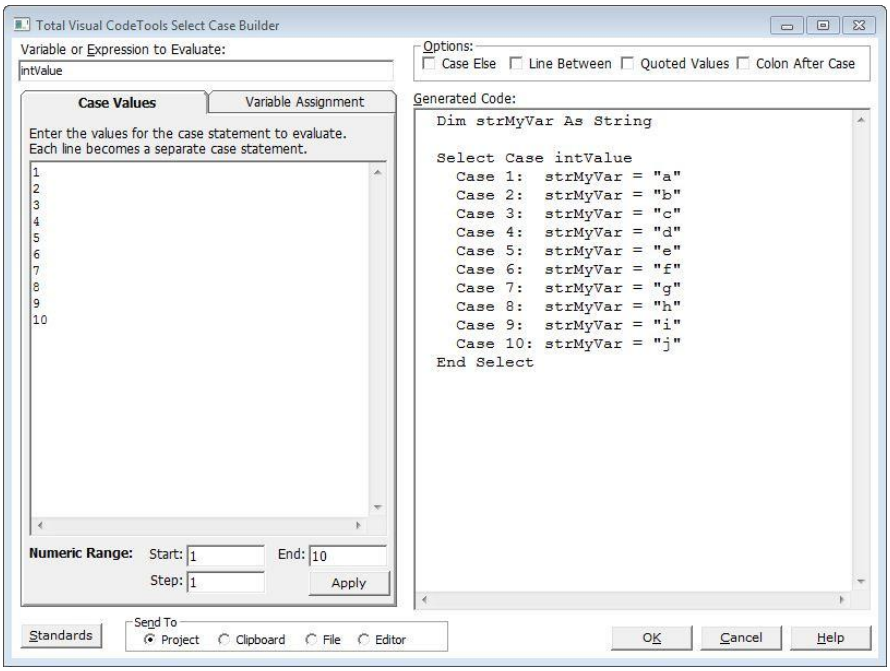
Of course there could be more lines after each Case line. Another benefit of a Case statement is that it can have multiple values or a range of values which eliminates the use of many OR statements. For instance, this would assign intResult to 1 if the value was either “Table” or “Query”:

```
  Case "Table", "Query":
    intResult = 1
```

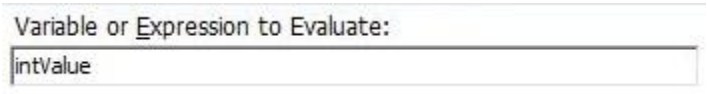
Look in your VB/VBA help file for complete SELECT CASE syntax options.

Invoking the Select Case Builder

The Select Case Builder simplifies the creation of a Select Case statement by letting you easily list the items you want to evaluate and assign a variable based on the value. Choose the “Select Case Builder” from the Total Visual CodeTools menu or toolbar:

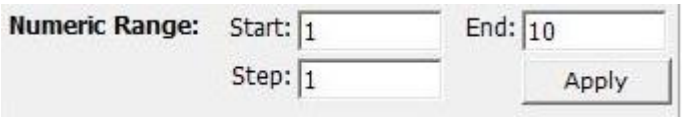


Enter the variable name or expression name to evaluate in the "Variable or Expression to Evaluate" text box.



Then on the Case Values tab, enter the individual values to evaluate. Each row becomes its own case statement. You can easily paste a block of text with multiple values.

Alternatively, you can generate a range of numeric values by specifying the starting and ending values, and the step between them, then press Apply:



Generate a Range of Numeric Values to Evaluate

The code in the "Generated Code" box automatically reflects your selections. The following options are available:

Case Else

Check this option to add a Case Else statement to the generated code.

Lines Between

Check this option to add blank lines between each Case statement.

Quoted Values

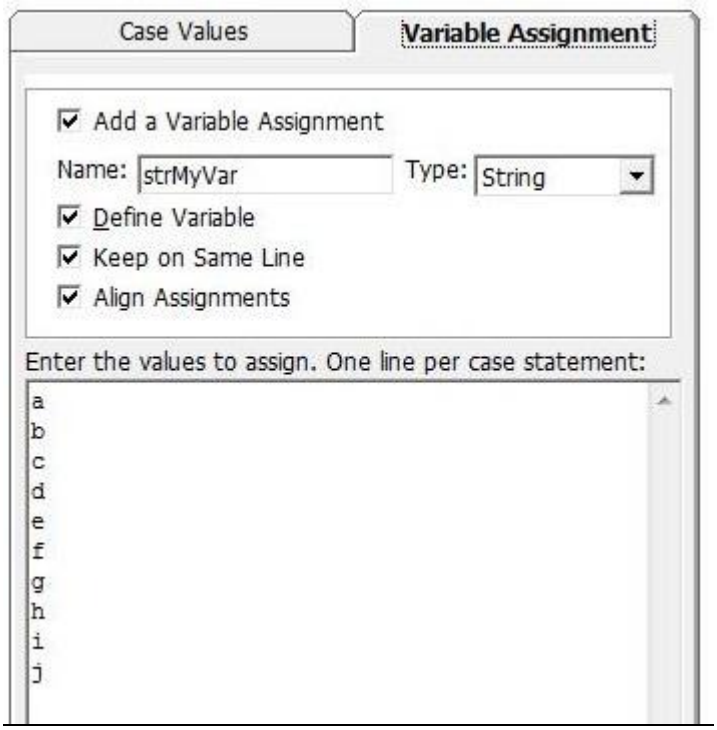
Check this option to add quotes around the values specified in the Case Value tab.

Colon After Case

Check this option to add colons to the end of each case statement.

Assign a New Variable

Often, the Select..Case syntax is used to assign a value for each Case statement. The Select Case Builder makes this easy under the Variable Assignment tab:



Variable Assignment Options

Values to Assign

Starting at the bottom, you can enter the value assigned for each Case statement in the [Enter the values to assign…] box.

Variable Name

Enter the name of the variable being assigned the value.

Variable Type

Specify the data type of the variable. If no values are provided in the [Enter the values to assign…] box, values are automatically generated.

If it's numeric, it is assigned a value from 1 to the number of items being evaluated. For other types, it's simply assigned a default value so you can update it once it's in your module. For instance, strings and variants are assigned to a zero length string "", booleans are True, dates to Now, etc.

Define Variable

Checking this adds a Dim statement with your variable name and type.

Keep on Same Line

If you select this option, the variable is kept on the same line as the case statement:

```
Case "Table": intResult = 1
```

If it's not checked, it's kept on a separate line:

```
Case "Table"
  intResult = 1
```

Align Assignments

If you've selected "Keep on Same Line", this option aligns all the variable assignment so they start on the same column. This makes it easier to see the values being assigned to the variable:

```
Select Case strValue
  Case "Customer":  intResult = 1
  Case "Sales":     intResult = 2
  Case "Employees": intResult = 3
End Select
```

Example of a SELECT..CASE Statement, Aligned

This is what it looks like without alignment:

```
Select Case strValue
  Case "Customer": intResult = 1
  Case "Sales": intResult = 2
  Case "Employees": intResult = 3
End Select
```

Example of a SELECT..CASE Statement, Not Aligned

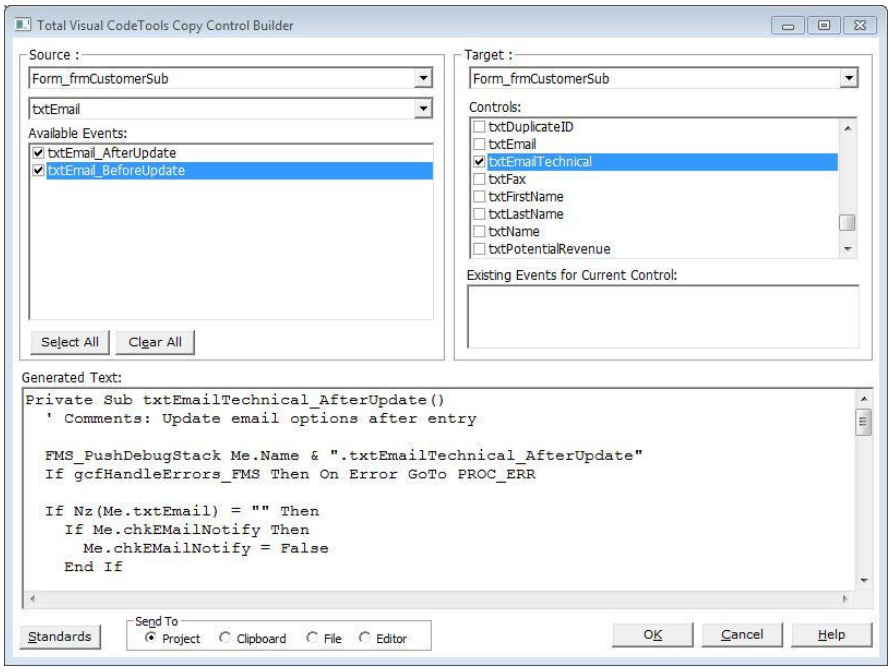
Copy Control Code Builder

When creating forms and reports for an application, you often find yourself using the same control code for multiple forms and reports. For instance, if every form has a [Close] button, chances are that the events for this button are similar if not identical. Other controls may have multiple event procedures that would be nice to copy all together or selectively.

The repetitive task of finding specific event code that you've already written and copying it can be tedious and error-prone, especially if you also have to rename the procedures for the new control name. The Copy Control Code builder helps you increase your productivity and accuracy by copying a control's code to another control on that object or another

Put Your Source and Target Objects in Design Mode

You must have both the source form/report and the target form/report open in design view before invoking this builder. Then, choose "Copy Control Code Builder" from the Total Visual CodeTools menu or toolbar.



Copy Control Code Builder

Specify the Source Object, Control, and Events

In the Source section (top left), select the form or report containing the control code to copy. The list shows the objects that are currently in design mode. Then select the control with the code to copy from. When you select the control, the list box displays the events that are available to copy—select the events to copy by checking the corresponding check boxes.

Specify the Target Object and Controls

After selecting the desired options in the Source section, use the Target section to specify where you want the code copied. First select the target form or report to show its list of controls. The list box displays existing events for the currently highlighted control. If the same event already exists for the control, you may not want to select it. Check the controls to copy the events.

Using the Generated Code

The new code is shown in the Generated text section. The name of the source control is replaced by the target control's name. The number of lines between procedures is set by your Code Cleanup standards.

Once the Copy Control Code Builder generates the code you want, select your "Send To" option and press [OK] (see **Using Generated Code** on page 71 for more information).

Note that if you choose Project, the code is inserted in your module when you invoked the builder, which may or may not be your Target module. In general, you should send this code to the Clipboard, go to the target module, then paste it where you'd like.

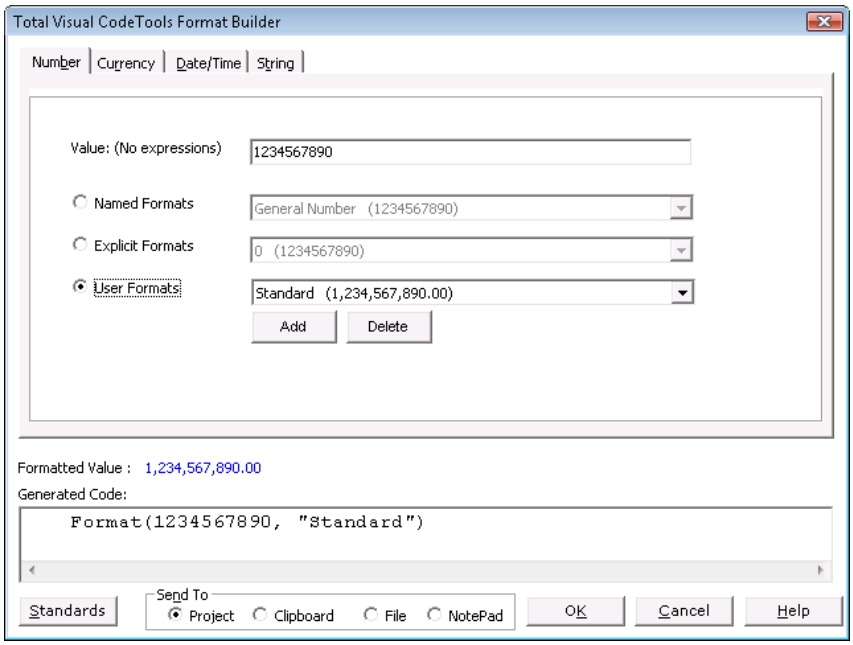
The Copy Control Code builder does not overwrite existing events. If you have an event with the same name that as your target, a second event is inserted, and you will need to review your code to determine if the old event should be renamed or removed.

Format Builder

The VB/VBA Format function allows you to customize the way values are displayed and printed, but it is difficult remember the arguments and syntax for this function, and the online help is very weak.

The Format Builder makes it simple to format and preview expressions according to common or custom date, number, string, and currency formats. You can even add your own formats to make it easy to reference them in the future.

To open this Builder, choose “Format Builder” from the Total Visual CodeTools menu or toolbar:



Format Builder

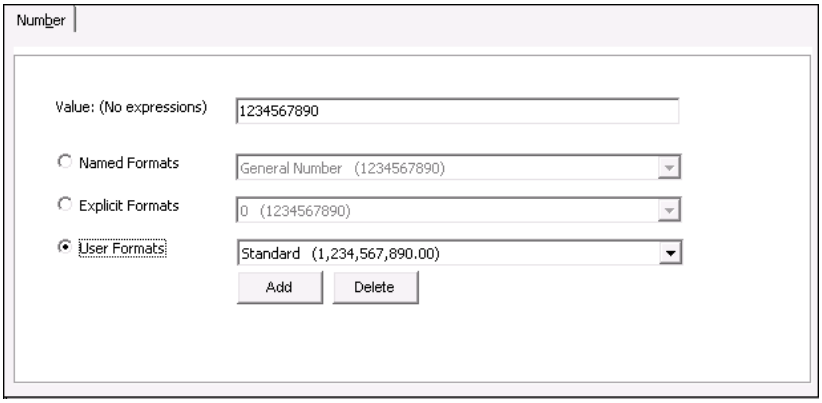
There are four tabs for the different types of formats to build: Number, Currency, Date/Time, and String. Each tab is similar. You enter a value for that data type, and select from Named, Explicit and User defined formats. The Named formats use the settings under the Windows Regional and Language Options set by the Control Panel.

As you select a tab and format, the formatted value is shown at the bottom along with the generated code.

The options available depend on the tab you select:

Number Tab

When you select the Number tab, these options are displayed:



Number Options

Value

Enter a numeric value to format.

Format Types

Select the "Format" option to use. The dropdown lists show the format of the value you entered for each option.

Named Formats

These are the named numeric formats such as General Number, Currency, Fixed, Standard, Percent, Scientific, True/False, and Yes/No. By using these, your format respects the user's Windows regional settings for things like the symbols for currency, decimals, and thousand separators.

Explicit Formats

Explicit formats use the raw syntax for specifying the format details for numbers. This includes sections for currency $, thousand separators, number of decimals, and formatting negative values differently.

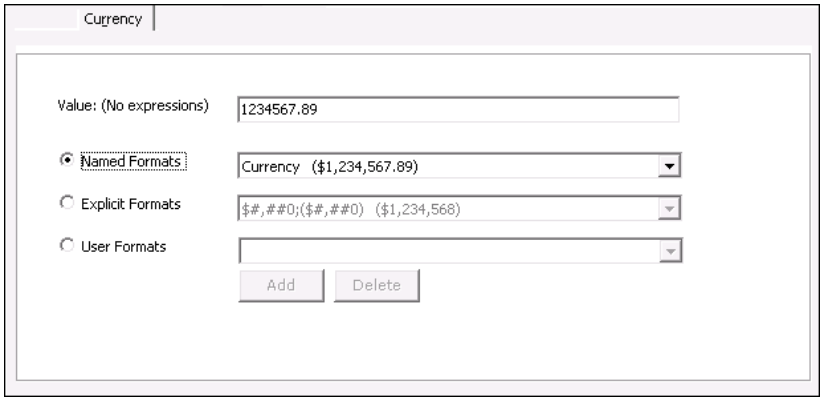
User Formats

Specify your own format by pressing the [Add] button and entering a new format. Note that these formats are saved locally and are not part of the global shared standards.

Select and press the [Delete] button to remove it from your list.

Currency Tab

When you select the Currency tab, these options are displayed:



Currency Options

Value

Enter a numeric value to format. Do not include a currency sign like $ or €.

Format Types

Select the "Format" option to use. The dropdown lists show the format of the value you entered for each option.

Named Formats

There is only one named Currency format which respects the user's Windows regional settings for currency. This is a big problem if your data is in $ but your user defaults to € and sees it that way. It is not a problem if you expect your users to provide their currency values and you want to simply display it.

Explicit Formats

Several explicit formats are provided for dollars, euros, sterling, yen, and Australian dollars.

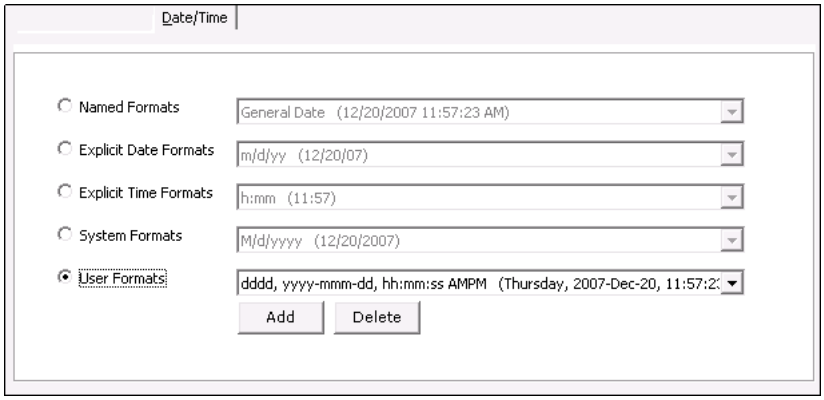
User Formats

Specify your own format by pressing the [Add] button and entering a new format. Note that these formats are saved locally and are not part of the global shared standards.

Select and press the [Delete] button to remove it from your list.

Date/Time Tab

When you select the Date/Time tab, these options are displayed:



Date/Time Options

Format Types

Select the "Format" option to use. The dropdown lists show the format of the value you entered for each option.

Named Formats

These are the Date Time formats named by Windows such as General Date, Long Date, Medium Date, Short Date, Long Time, Medium Time, Short Time. By using these, your format respects the user's Windows regional date time settings including month day order.

Explicit Date Formats

This provides explicit date time symbols so your dates are always shown the same way regardless of the user's regional settings. However, if the name of the month (mmm or mmmm) or day of week (dddd) is referenced, it reflects the language set by Windows.

Explicit Time Formats

This lets you create Time only formats, 24 hour clocks, AM/PM and combinations of date and time formats.

System Formats

This is the list of explicit date and time formats that Windows provides. There will be some overlap between the items here and the items in the Explicit Date and Time lists.

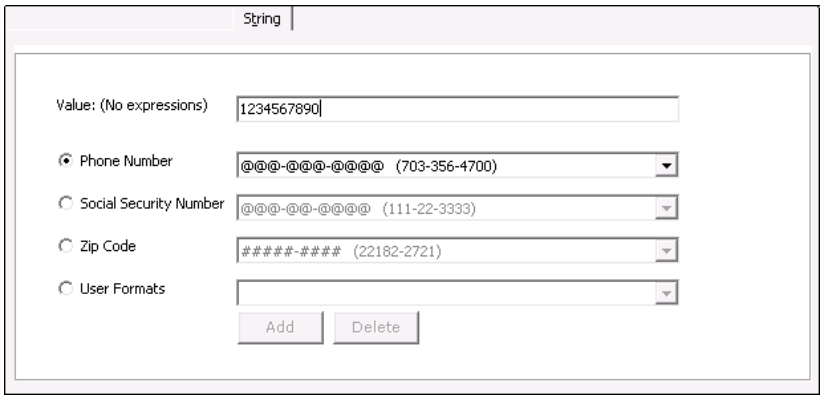
User Formats

Specify your own format by pressing the [Add] button and entering a new format. Note that these formats are saved locally and are not part of the global shared standards.

Select and press the [Delete] button to remove it from your list.

String Options

When you select String, the following options are displayed:



String Options

Value

Enter a string value to format. Unlike the other tabs where a number can apply to multiple formats, you may need to change this value depending on the format you select.

Format Types

Select the “Format” option to use. The dropdown lists show the format of the value you entered for each option.

- Phone Number
- Social Security number
- Zip Code

User Formats

Specify your own format by pressing the [Add] button and entering a new format. Note that these formats are saved locally and are not part of the global shared standards.

Select and press the [Delete] button to remove it from your list.

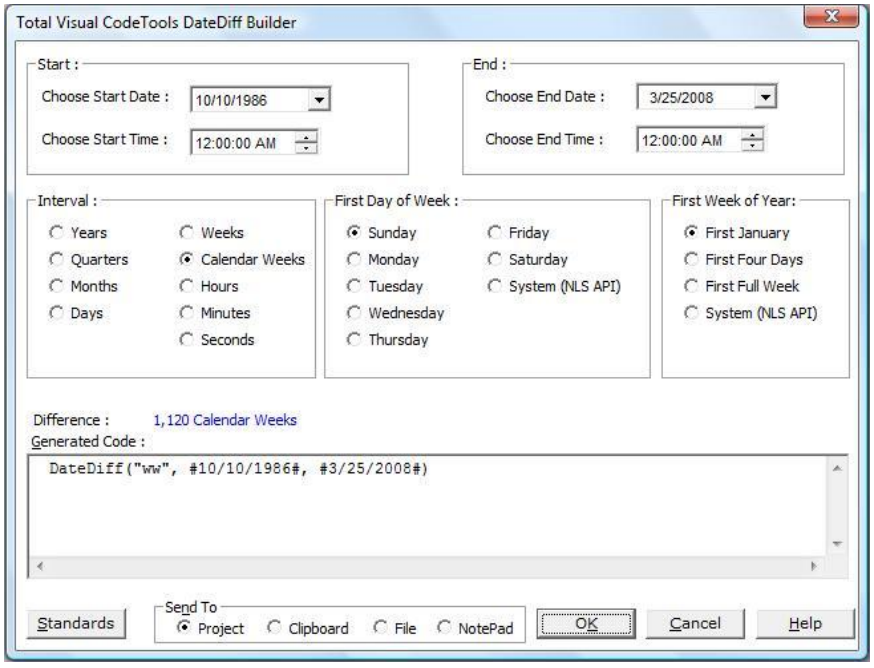
Using the Generated Code

Once the Format Builder generates the code you want, select your "Send To" option and press [OK] (see **Using Generated Code** on page 71 for more information).

DateDiff Builder

The VB/VBA DateDiff function calculates the time interval between two specified dates/times, but it is difficult to remember the syntax and arguments of the function. The DateDiff Builder builds code to calculate date and time differences, including years, quarters, months, weeks, days, weekdays, hours, minutes, seconds, and more.

To open this Builder, choose "DateDiff Builder" from the Total Visual CodeTools menu or toolbar:



DateDiff Builder

The code that is generated calculates the difference between the start date/time and the end date/time. The following options are available:

Start Date/Time

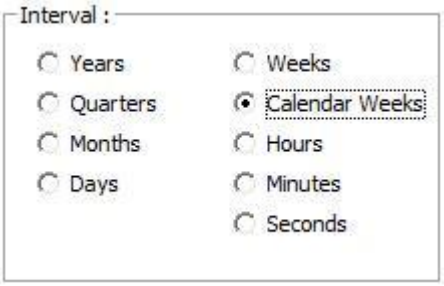
Select the start date and time. If you don't want to include the time, enter 12:00:00 AM for the time.

End Date/Time

Select the end date and time. If the end date is before the start date, the code returns a negative number.

Interval

Specify the interval of time to use in the calculation: Years, Quarters, Months, Days, Weeks, Calendar Weeks, Hours, Minutes, or Seconds.



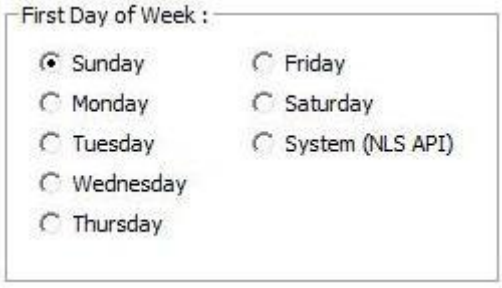
The DateDiff function calculates the number of intervals between two dates. When using the DateDiff function, please be aware of the following:

- If you select the interval “Weeks,” the code returns the number of weeks between the two dates. If the start date falls on a Monday, the DateDiff function counts the number of Mondays until the end date. If you select the interval “Week,” the function returns the number of calendar weeks between the dates (inclusive of the end date, but not the start date).
- When comparing December 31 to January 1 of the immediately succeeding year, the Year interval returns 1 even though only a day has elapsed.

Refer to your VB/VBA documentation for more information about the DateDiff function.

First Day of Week

This option is enabled if the “Weeks” or the “Calendar Weeks” interval is selected. Specify the first day of the week to consider in the calculation.



VB/VBA help refers to the Weeks option ("w") as Weekday but it really calculates the number of weeks, not days between the dates. Here is the difference between Weeks and Calendar Weeks ("ww"), which VB/VBA help calls "week":

- The Weeks option ("w") calculates the number of days between the two dates and is one for every seven days regardless of the first day of week.
- The Calendar Weeks option ("ww") calculates the number of First Day of Weeks that appears between the two dates. So if the two dates are on Friday and the following Monday, and the First Day of Week is Sunday, there is one Calendar Week difference. The Weeks option would be zero because it's less than 7 days.

If the start date falls on the First Day of Week, a week is completed on that day the next week.

First Week of Year

Specify the option to use to determine the first week of the year:

- **First January:** Start with the week that contains January 1 (default).
- **First Four Days:** Start with the first week containing at least four days.
- **First Full Week:** Start with the first full week of the year.
- **System:** Use the National language support API package to determine the first week of the year.

Using the Generated Code

As you change the options the difference is shown so you can verify it's returning what you expect.

Once you are ready, select your "Send To" option and press [OK] (see **Using Generated Code** on page 71 for more information).

Chapter 5: Unused Variable Analysis

As one writes code, variables are often created but not used or old code deleted without deleting the corresponding variable declaration. Unused Variable Analysis helps you detect and eliminate unnecessary code from your project. Not only does it make your code more compact, the analysis may highlight problems where you thought the code should be doing or tracking something that it is not.

Topics in this Chapter

- **Unused Variable Overview**
- **Running Unused Variable Analysis**
- **Unused Variable Analysis Limitations**
- **Using the Results**

Unused Variable Overview

As one writes code, variables are often created but not used or old code deleted without deleting the corresponding variable declaration. Total Visual CodeTools' Unused Variable Analysis helps you detect and eliminate unnecessary code from your project. Not only does it make your code more compact, the analysis may highlight problems where you thought the code should be doing or tracking something that it is not.

The Unused Variable feature finds more than just variables:

- Variables defined with Dim, Public, Private, Static, etc.
- Procedure parameters
- Constants
- Classes that are not referenced
- User defined types (Type..End Type syntax)
- User defined type elements

The Unused Variable Analysis is based on the scope you specify: one procedure, one module, multiple modules, or the entire project.

Is it Important to Eliminate Unused Variables?

Getting rid of unused variables may seem like a waste of time. If it's unused, why bother, since it's not hurting anything? In a world with pressure to just get code to work, rather than "perfection", it may seem unnecessary to worry about unused variables.

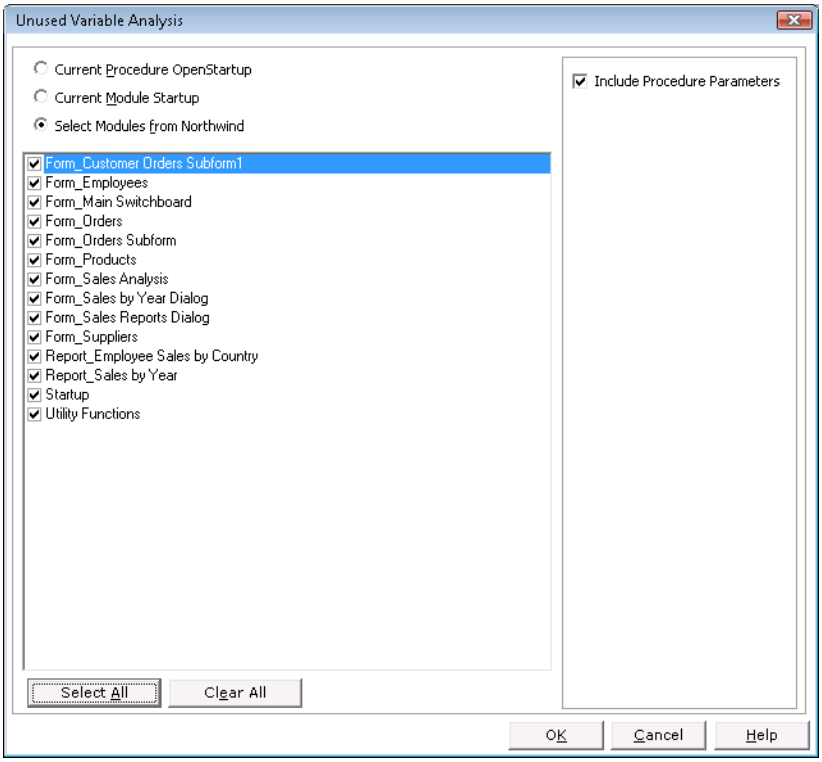
However, reviewing and eliminating unused variables can be helpful for many reasons:

- Eliminating unnecessary code makes your remaining code smaller, easier to maintain, and more readable
- Unused variables may indicate bigger problems. For instance, you created a variable and intended for it to do something but never implemented it. Discovering that it's unused may require a fix to implement it rather than delete the variable.
- Unused procedure parameters (excluding mandatory parameters for event procedures) often indicate a bug. The procedure is allowing different values to be passed to the parameter, but its behavior never changes. Either get rid of the parameter or have it do what it's supposed to do.

By running the Unused Variable Analysis feature on a regular basis, you'll keep your code cleaner and apply Best Practices to your coding skills.

Running Unused Variable Analysis

Select “Unused Variables” from the Total Visual CodeTools menu or toolbar:



Unused Variable Analysis Selections

From this form, you can select what objects to analyze. You can select the current procedure, current module, or individual modules. Press [Select All] to select all the modules.

Include Procedure Parameters

There is a check box to include or exclude procedure parameters. Procedure parameters are the variables passed into a procedure:

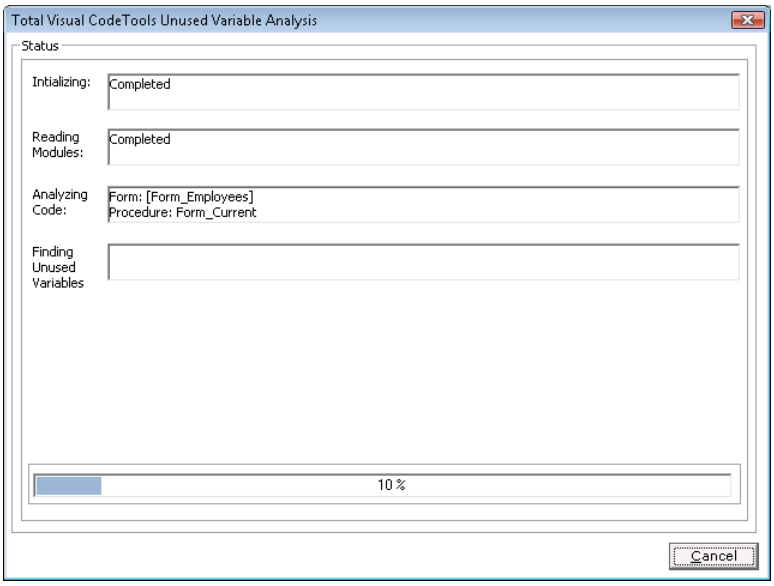
```
Function Sample(Parameter1 As String)
```

Built-in event procedure on forms and Access reports, can generate a lot of unused parameters because one often ignores those parameters. This option is provided if the list of unused procedure parameters becomes overwhelming and you’d prefer to exclude them from the results.

However, keep in mind that finding unused parameters in your procedures is very important since it indicates the procedure is not doing anything with the value that is passed to that parameter which may be a bug.

Generate Results

When you press [OK], Total Visual CodeTools starts its code analysis and presents this status screen:



Unused Variable Analysis Status

Unused Variable Analysis goes through all your selected modules and determines all the variables that are defined. It also determines all the uses of the variables. By eliminating the ones that are used, the list of unused variables is generated. When it's completed, a message box appears and you can view your results.

Unused Variable Analysis Limitations

If you do not select all the modules in your project, you run the risk of incorrectly flagging unused variables that are used in the modules you did not select. Only the modules you select are used in the analysis.

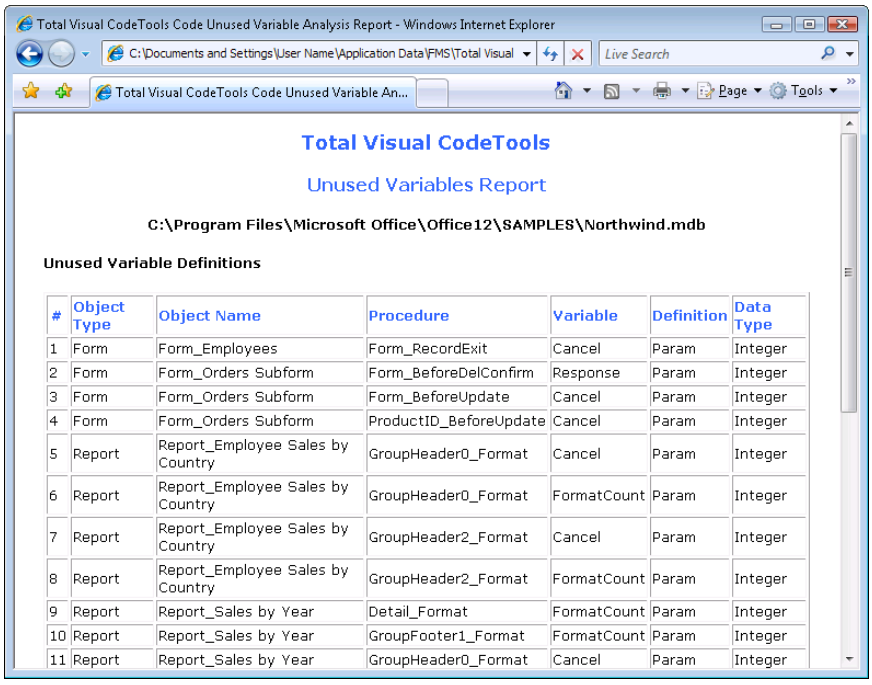
If your variables are global (public) and referenced by modules that were not selected, they may be flagged as unused when they are actually used.

Unused variables defined in a procedure are always properly flagged. Private module level variables are also accurately documented as unused if the module was selected. It's the global ones that won't be completely accurate unless all objects are selected.

If the project is used as a library and referenced external to this project, those references are obviously not considered, so you'll need to be careful before deleting any unused global variables or classes.

Using the Results

The results are presented in an HTML file called Unused.htm in your Application Data folder. It is opened with your default Internet browser:



#	Object Type	Object Name	Procedure	Variable	Definition	Data Type
1	Form	Form_Employees	Form_RecordExit	Cancel	Param	Integer
2	Form	Form_Orders Subform	Form_BeforeDelConfirm	Response	Param	Integer
3	Form	Form_Orders Subform	Form_BeforeUpdate	Cancel	Param	Integer
4	Form	Form_Orders Subform	ProductID_BeforeUpdate	Cancel	Param	Integer
5	Report	Report_Employee Sales by Country	GroupHeader0_Format	Cancel	Param	Integer
6	Report	Report_Employee Sales by Country	GroupHeader0_Format	FormatCount	Param	Integer
7	Report	Report_Employee Sales by Country	GroupHeader2_Format	Cancel	Param	Integer
8	Report	Report_Employee Sales by Country	GroupHeader2_Format	FormatCount	Param	Integer
9	Report	Report_Sales by Year	Detail_Format	FormatCount	Param	Integer
10	Report	Report_Sales by Year	GroupFooter1_Format	FormatCount	Param	Integer
11	Report	Report_Sales by Year	GroupHeader0_Format	Cancel	Param	Integer

The Unused.htm file is organized into these sections:

- Unused Variable Definitions
- Unused Classes
- Unused Types
- Unused Type Elements

Unused Variable Definitions

The entire list of unused variables is documented. The list is numbered and sorted by object type, object name, procedure, and variable name (could also be a constant). For each variable, its definition (Dim, Public, Private, Const, etc.) is listed. If it's a procedure parameter, the Definition is Param. The data type, if defined, is also shown.

Assuming you understand the limitations of the analysis and what you selected, go down this list and edit your modules to eliminate these unused variables.

You may find that you do not want to delete some unused code such as constants that are part of Windows API values since you may want to keep them around so that Intellisense displays them when selecting options for the API calls.

Unused Classes

Classes (*.cls files) that are defined but not referenced. An unused class can be deleted, but since a class usually contains lots of code, you should be careful before doing this. If this project is used as a library, the class may be used externally, so deleting it would be wrong.

Unused Types

The Type..End Type syntax lets you create user defined types to hold multiple related values. Variables are then created based on the type:

```
Type Customer
  ID As Long
  Name As String
End Type

Private mtypCustomer As Customer
```

If the type is never referenced as a variable or passed as a procedure parameter, it is unused. Verify it is not used by searching it across your project. If so, it would be safe to delete.

Unused Type Elements

Type elements are the items defined inside the Type..End Type statement (e.g. ID and Name in the previous example). Individual elements may be unused even if the type is used, so you may want to delete them.

However, you should not delete these if the type variable is used as a parameter for something like a Windows API call since we don't document the internals of the API call and cannot determine which element is used or not. If the Type is only used by your code, you may want to delete it if it's never assigned a value or referenced.

Chapter 6: Additional Tools

In addition to Builders, Unused Variables, Code Cleanup, and Code Delivery, Total Visual CodeTools offers several other tools to help you work more efficiently. This chapter discusses these tools.

Topics in this Chapter

- **Macro Recorder**
- **Close Code Windows**
- **Clear Immediate Window**
- **Block Commenter**
- **VBE Color Schemes**

Overview of the Available Tools

Total Visual CodeTools offers several additional tools to ease the development process. The following tools are available from Total Visual CodeTools menu and toolbar:

Macro Recorder

The **Macro Recorder** utility increases your productivity by letting you record your keystrokes and replay them to eliminate repetitive typing.

Close Code Windows

The **Close Code Windows** tool makes it easy to clean up your development environment by closing all open code windows.

Clear Immediate Window

The **Clear Immediate Window** tool makes it simple to clear the content of the Debug/Immediate window.

Block Commenter

The **Block Commenter** utility provides a quick and easy way to comment out text and document the date, time, and developer.

VBE Color Schemes

The **VBE Color Schemes** tool helps you visually organize the appearance of your Visual Basic Editor.

Macro Recorder

While writing your code, you may find yourself repeating the same tasks and keystrokes multiple times. This is not only tedious, but error prone. The Macro Recorder utility increases your productivity and accuracy by recording and replaying your keystrokes to eliminate repetitive typing.

Press the  button or select “Macro Recorder” from the menu to display the Macro Recorder toolbar:

Macro Recorder Toolbar

It's very simple to record and replay your keystrokes:

- Click the Record  button to start recording.
- Edit your code and move around using the keyboard. Avoid using your mouse and selecting menu items. If you want to cut, copy or paste, use the [Ctrl] X, C, or V key combinations.
- Click the Stop  button to finish recording.
- Move your cursor to where you'd like to replay your keystrokes
- Click the Play  button to replay your keystrokes. You can press the play button as many times as you'd like, but be sure it's working the first time.
- Click the Help button to open the Help topic for the Macro Recorder

Macro Recorder Limitations

The Macro Recorder tool can save you time and frustration by automating repetitive tasks, but there are some limitations:

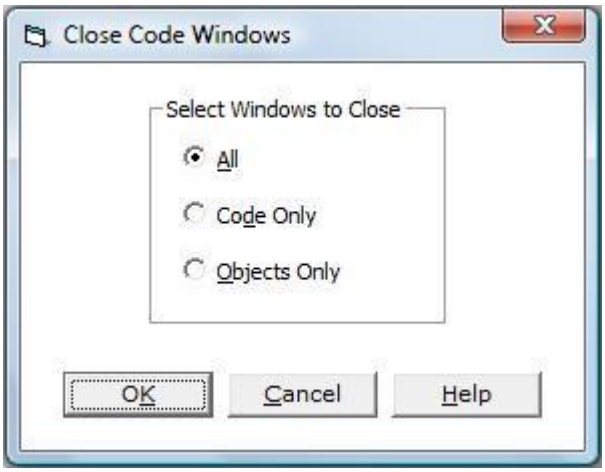
- The Macro Recorder only records and replays keystrokes in your current code window. That means if you move to another module or select items from the menu, those actions are not recorded.
- The Macro Recorder does not record mouse movements and mouse clicks, only keystrokes. While recording a macro, use the keyboard rather than the mouse to change cursor location in the code window.
- Menu selections via the mouse or [Alt] key combinations are not recorded. The [Esc] key is also not recorded.
- [Ctrl] key combinations are handled for the clipboard (X, C, and V for cut, copy, and paste) are direction keys in the editor. Other [Ctrl] key combinations for menu selections may be "absorbed" by the menu and are therefore not recorded.
- With the exception of [F1], function keys cannot be recorded.
- To avoid undesired results, always test macros before running them repeatedly.

Close Code Windows

Sometimes after opening multiple objects and modules, the IDE gets cluttered with many code windows. To quickly get rid of the confusion, this tool closes all open windows.

Instead of closing each window in turn, simply click the button from the Total Visual CodeTools toolbar, or select “Close Code Windows” from the Total Visual CodeTools menu.

In Visual Basic, form objects and code are opened separately. If you have a combination of objects and code open, a prompt allows you to choose all or just code or objects to close:



Close Code Window Options for Visual Basic

This option is not available in Access because the form and report objects are open in the database and not the VBA editor.

Clear Immediate Window

Usually in the development process, the Debug window (or the Immediate window) gets filled with output statements. To get rid of the annoying task of clearing it manually, just run this tool.

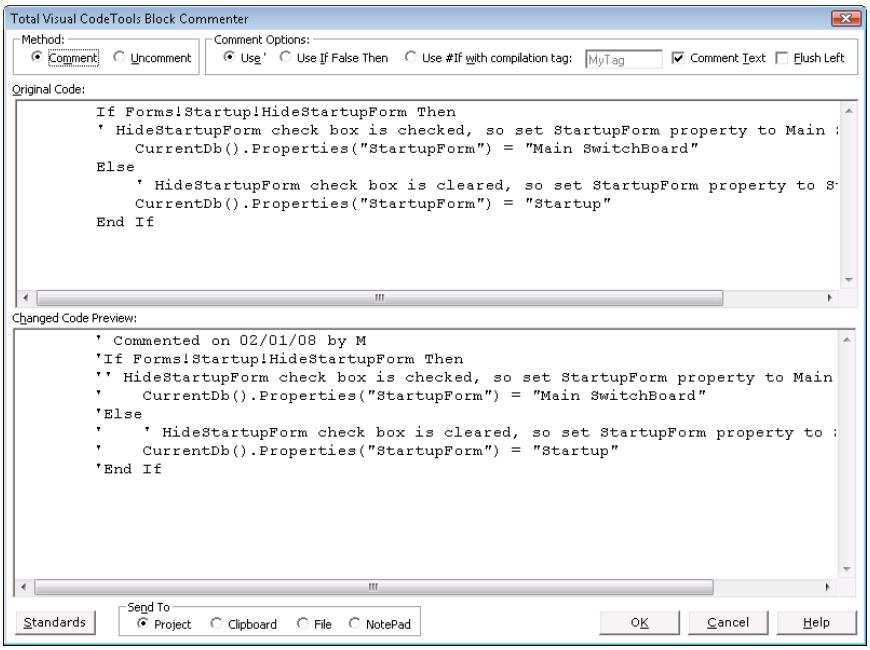
Click the button from the Total Visual CodeTools toolbar, or select “Clear Immediate Windows” from the Total Visual CodeTools menu.

Block Commenter

Often in the development cycle, you need to comment a block of code. The Visual Basic Editor has a built-in “Comment Block” feature that simply comments or uncomments a selected block of text, but you may want additional features. The Total Visual CodeTools Block Commenter feature lets you:

- Comment out text and add a custom comment to include the date, time, and developer that did it
- Put the block in an If False Then..End If block
- Use a compiler directive #If..#End If block
- Comment the lines and remove all indentations (flush left)
- Remove comments

Highlight the text you want to modify in the editor. Then click the  button from the Total Visual CodeTools toolbar, or select “Block Commenter” from the Total Visual CodeTools menu:



Block Commenter

Your selected code appears in the Original Code section. The new code is in the lower Changed Code Preview section.

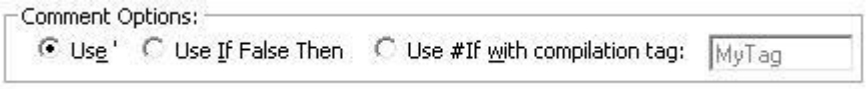
Method

Choose whether to add comments or remove them:

- Comment adds an apostrophe ' at the beginning of each line (if the Use ' option is selected)
- Uncomment removes the apostrophe at the beginning of a line.

Comment Options

For the Comment method, choose among three options:



Use '

This adds an apostrophe at the beginning of each line

Use If False Then

Select this option if you want to disable code by surrounding it with an "If False Then….End If" block:

```
If False Then
  <your highlighted code>
End If
```

The code is automatically indented within the IF..End If block. This effectively disables the code without using comments.

Use #If with Compilation Tag

Select this option if you want to disable code by surrounding it with an "#If Tag Then….#End If" clause and specifying a compiler directive tag when the project is compiled.

```
#If MyTag Then
  <your highlighted code>
#End If
```

Compiler directives let you optionally block out the code. This is a common practice in Visual Basic, but rarely used in VBA.

Additional Comment Options

Two additional options are available for the Comment method:

Comment Text

This adds a comment line above your comment block such as:

```
' Commented on 02/26 by MP
```

This is helpful for documenting when your change was made. The text is customizable and can include the current date, time, and developer name or initials. Set this on the Standards, Commenting, General Tab.

Flush Left

Select this option to remove all the indentations and push all the commented lines to the left.

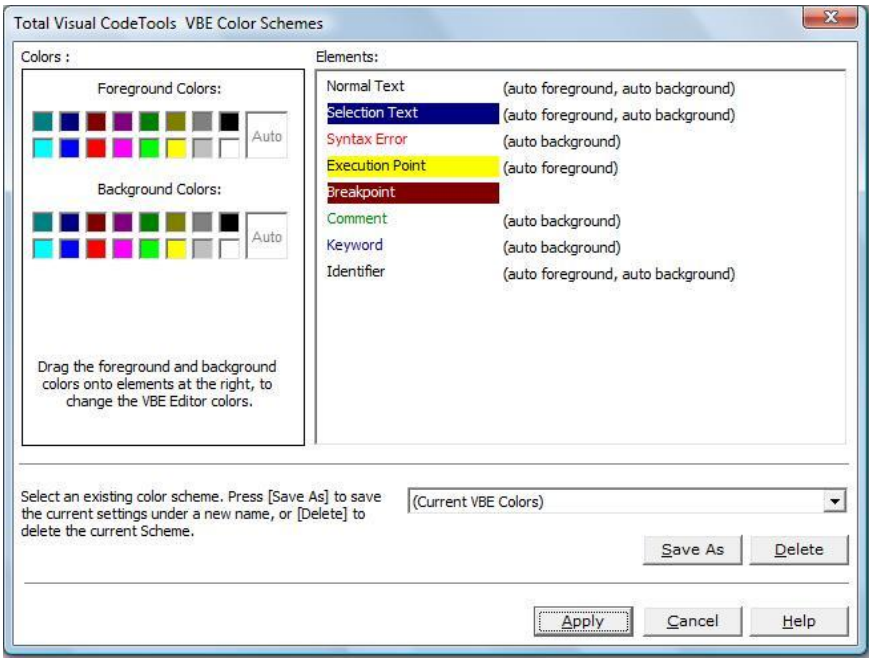
Using the Generated Code

Once the Block Commenter generates the code you need, select Project in the Send To section, and press [OK] to have it replace your existing code (see **Using Generated Code** on page 71 for more information).

VBE Color Schemes

The Visual Basic Editor allows you to change its color schemes through the Tools, Options menu. This is somewhat awkward, since there is no way to look at the appearance of the editor with all your selected colors at the same time. Additionally, there is no method to store multiple color schemes.

Click the  button on the Total Visual CodeTools toolbar, or select “VBE Color Schemes” from the menu:



VBE Color Schemes

Using VBE Color Schemes

Choose the foreground and background colors you need for your editor by dragging and dropping the colors on to the elements in the list.

After you are satisfied with the appearance, you can save the color schema.

Applying the Color Schema

Apply the saved color scheme to the editor by pressing [Apply]. The results, however, will not be reflected until you close and start the editor again.

To return to the default color schemes, select the “Default VBE Colors” item from the list, and restart your code editor.

Chapter 7: Code Cleanup

Code Cleanup is one of the most powerful features of Total Visual CodeTools. Code Cleanup standardizes the code in your project to your specifications. It adds error handling to procedures without it, standardizes code indentations, applies your variable naming convention, inserts comment structures, sorts procedures, and more! This tool is ideal when you inherit someone else's code, or when you need to standardize code within a development team. The result is code that is more readable, robust, and maintainable.

Topics in this Chapter

- **Code Cleanup Overview**
- **Operate from Backups**
- **Ensure the Integrity of Your Project**
- **Running Code Cleanup**
- **Code Cleanup Options**
- **Code Cleanup Processing**
- **View Messages**
- **Preview the Code**
- **Apply the Changes**

Code Cleanup Overview

As a software developer, chances are you occasionally inherit code that does not meet your coding standards. This makes it very difficult even to read, much less to maintain and enhance. Given tight deadlines and time constraints, even your own code may have inconsistent indentation, naming conventions, and comment structures, or procedures without error handling and other problems. Trying to fix this manually is an extremely time consuming and error prone process.

The Code Cleanup tool addresses these problems by applying consistent formatting, adding error handling to procedures that lack it, and more.

Preparing for Code Cleanup

Because Code Cleanup can make a massive number of changes to your code, it is extremely important that you prepare for the process correctly. The following sections outline the steps to take before running Cleanup:

- Operate from Backups
- Ensure the Integrity of Your Project
  - Ensure Exclusive Access
  - Compile All Code
  - Check Out Visual SourceSafe Objects

Operate from Backups

If you are planning to use Code Cleanup to modify your code, it is very important that you make a backup of your project in case the Cleanup process is interrupted or fails.

How much preparation work you do depends on the number of changes you are asking Code Cleanup to do for you. For example, if you are using Code Cleanup to modify just one procedure, you don't need to have a backup of your entire project. However, if you are working on multiple objects, or especially the entire project, you need a backup of your project, in case you need to roll back the changes. Once Code Cleanup applies its changes, it may not be possible to undo.

Important!
You must create a backup of your code before running any Code modification tools. FMS cannot provide technical support if you do not have a backup of your code.

Backing up your Visual Basic Project

Microsoft Visual Basic stores each component of your project in separate files at the operating system level. Typically, all files are stored in one directory, but there may be more than one, especially if you're sharing code across projects. In order to make a backup, use Windows Explorer to make a copy of all your objects into a new directory. For help, consult your Visual Basic Project file (the .VBP file) for a list of all objects referred to by the project. The following sample VBP file shows where to look for file dependencies.

```
Type=Exe
Reference=*\G{00020430-0000-0000-C000-
000000000046}#2.0#0#C:\WINDOWS\SYSTEM\StdOle2.Tlb#OLE
Automation
...

Object Information: The VBP file contains an entry for each
class, module, form, designer, etc. in your project. If the
entry doesn't have a path, the object is stored in the same
directory as the VBP file.

Class=CMyClass; CMyClass.cls
Class=CSharedUtils; T:\Projects\SuperApp\CSharedUtils.cls
Module=modUtility; modUtility.bas
Form=frmMenu.frm
Form=T:\Project\SuperApp\SharedForms\frmAbout.frm
ResFile32="MyApp.RES"
```

Header Information

The header contains all references in your project. You do not need to make backups of these files since Total Visual CodeTools does not modify them.

Backing up Office Projects

Office applications that support VBA also support user forms and other code-related objects. In most cases, these objects are stored in the database or other document in which you are working. For example, in Microsoft Word, Microsoft Excel, and Microsoft PowerPoint, the associated .doc, .xls, or .ppt file contains all code objects. In order to create a backup of these projects, simply create a copy of the file using Windows Explorer. Other applications, such as Microsoft Outlook and Microsoft FrontPage, store their files in an external location.

The following list contains details for each VBA-enabled Microsoft Office application:

Microsoft Access

Microsoft Access stores all your VBA project information in the database.

Before making a backup, it is a good idea to run the Access "Compact and Repair Database" operation. This ensures that there is no corruption in your database. It also reclaims wasted space, allowing database operations to work faster.

Microsoft Word

Microsoft Word stores all VBA project information the Word document.

Microsoft Excel

Microsoft Excel stores all VBA project information in its spreadsheet document.

Microsoft PowerPoint

Microsoft PowerPoint stores all VBA project information in its PowerPoint document.

Microsoft Outlook

Microsoft Outlook stores your entire VBA project in a structured storage file with the extension of .OTM. The default location for this file is your \Windows\Application Data\Microsoft\Outlook directory.

Microsoft FrontPage

Microsoft FrontPage stores your entire VBA project in a structured storage file with the extension of .FPM. The default location for this file is your \WINDOWS\Application Data\Microsoft\FrontPage\Macros

Total Visual Agent, also from FMS, can help you with the process of creating and maintaining regular backup sets of Access databases and other files. For more information and a demo, visit www.fmsinc.com.

Ensure the Integrity of Your Project

Because Code Cleanup modifies your module code, there are several additional issues to keep in mind.

Ensure Exclusive Access

Make sure no one else is editing the project when you run Code Cleanup.

Compile All Code

To modify your modules, Total Visual CodeTools parses your code. It steps through each line of code and breaks it into its constituent parts: reserved words, variables, expressions, comments, etc. If your code has syntax problems and does not compile, Total Visual CodeTools may fail to parse your code properly, and may stop running or create erroneous modified code. To ensure this does not happen, compile all your modules and fix any syntax problems.

In VB, select Run, Start with Full Compile to verify it works. In VBA, choose Debug, Compile.. to compile your code.

Once your code is free of syntax errors and compiles, it is ready for Code Cleanup. If you made any changes to your code to correct syntax errors, you should make another backup of your code at this point to ensure your changes are safe.

Check Out Visual SourceSafe Objects

If your project is under Microsoft Visual SourceSafe (VSS) control (Microsoft's source code check-in/check-out version control program), you must check out all the modules before running Code Cleanup. Otherwise, Total Visual CodeTools cannot update your modules with the revised code.

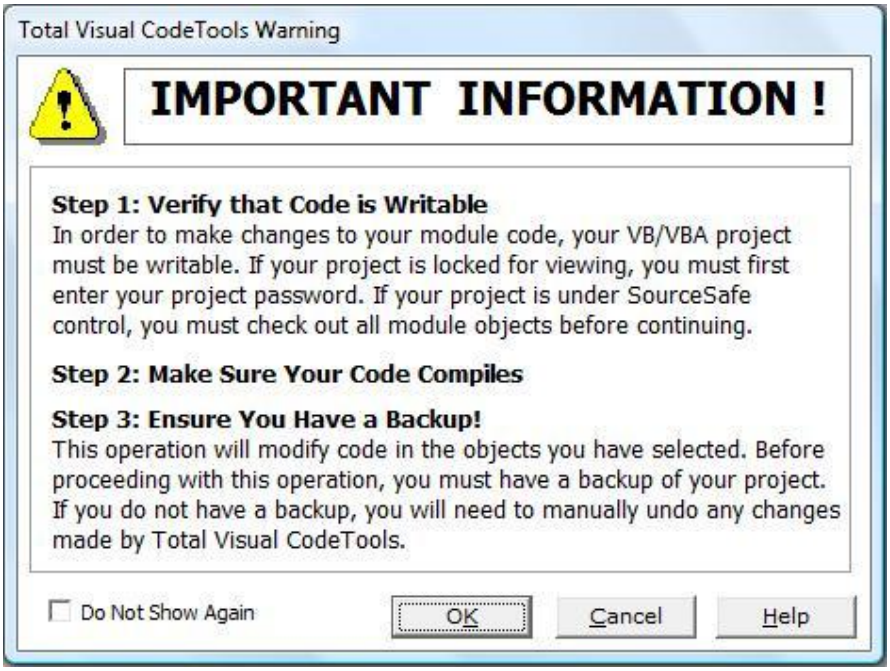
Running Code Cleanup

Once the preparatory steps are completed, you are ready to run Code Cleanup. Select "Code Cleanup" from the Total Visual CodeTools menu or toolbar. There are the steps:

- A warning message to make sure your project is prepared
- Selecting the objects for cleanup (current procedure, current module, or some or all modules)
- Specifying the cleanup options
- The Code Cleanup process runs to present you with the changes to be made
- The ability to preview the changes before applying them (canceling at this point leaves your code unmodified)
- Applying the changes to your project

Warning Message

When you start Code Cleanup, a warning message appears:



Code Cleanup Startup Warning

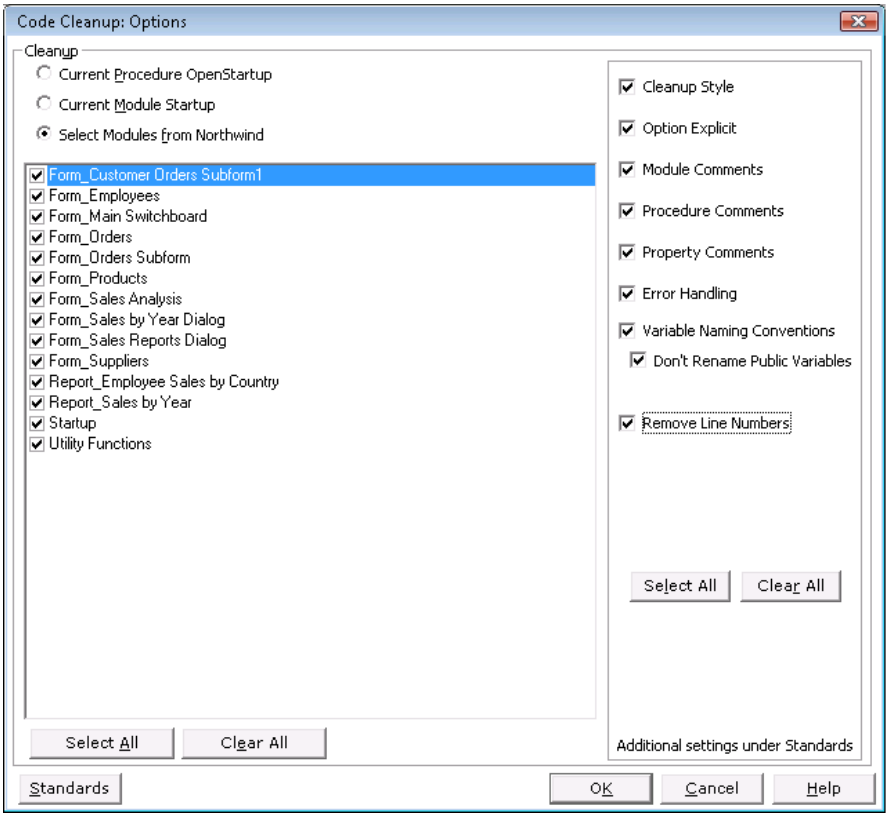
Make sure your code is backed up, compiled, and updateable (checked out of Visual SourceSafe if you're using that). Press [OK] to continue. Otherwise, press [Cancel] and address these issues.

Remember: Code Cleanup cannot recreate your original code after changing it, so a backup is very important.

If you do not want this warning page to appear again, check the Do Not Show Again box. This option is not available if the Standards file has a password, and you'll need to set this on the Standards Cleanup page.

Selecting Objects

Choose the objects to clean and how to clean them:



Code Cleanup Options

The left side of the form lets you specify the objects to modify. If you have the editor open and the cursor in a procedure, you can clean just that procedure, just that module, or any or all the modules in your project.

To clean all the code, simply choose the third option and press the [Select All] button.

Code Cleanup Options

The right side of the form lets you specify which cleanup routines you want to run on your code. The following options are available:

Cleanup Style

Standardizing the visual appearance of the code is arguably the most important factor for code readability. Choosing the "Cleanup Style" option tells Code Cleanup standardize code indentation, split single line If statements and statements separated by a colon, remove extra blank lines, sort the procedures, and more.

Style options are set under Standards. See **Cleanup Style** on page 29 for details.

Code Cleanup Options

Option Explicit

This feature inserts the "Option Explicit" statement into the declarations section of every module without it. This option is not available if you only selected the current procedure for cleanup.

One of the cardinal rules of writing reliable and maintainable code is to explicitly declare all of your variables. By putting this statement in your module, you are requiring that every variable in that module be explicitly declared using statements such as Dim, Const, Public, etc. See page 160 for information on why using "Option Explicit" is important.

While this statement is important to include in all modules, you should note that inserting Option Explicit into modules often causes the module to generate compilation errors if your code contains undeclared variables or there are typos with variable names.

If you think you have modules without the "Option Explicit" statement, you should select only this option, perform the cleanup process, try to compile your database, and fix any compilation problems. After your code compiles properly, use Code Cleanup for other tasks such as applying variable naming conventions.

Module Comments

The Module Comments feature adds a standard comment header to the top of each module. This may include project name, creation dates,

copyright notices, and ownership rights. You can also add a complete list of all procedure names and definitions. This feature is not available if you only selected the current procedure for cleanup.

Module Comments are customized under Standards. See **Code Cleanup: Module Comments (Cleanup Mod tab)** on page 36 for details.

Procedure Comments

The Procedure Comments feature adds comment headers to each procedure (sub or function). Although the program cannot determine what the code is actually doing and write the comments for you, it can take much of the drudgery out of the process of creating consistent comment headers for each procedure.

Procedure Comments are customized under Standards. See **Code Cleanup: Procedure and Property Comments (Cleanup Proc and Cleanup Prop tabs)** on page 40 for details.

Property Comments

The Property Comments feature inserts the specified property comment template into all property procedures.

Property Comments are customized under Standards. See **Code Cleanup: Procedure and Property Comments (Cleanup Proc and Cleanup Prop tabs)** on page 40 for details.

Error Handling

Error handling is one of the key attributes of robust applications. The Error Handling feature inserts error handling code into all procedures without it, and allows you to specify separate error handling routines for regular modules vs. class modules. Error handling tags can also be added in so that Total Visual CodeTools can automatically update the code in the future.

Total Visual CodeTools does not add error handling to every procedures. If it did, it would conflict with your existing error handling routines. Rather, error handling is only added to procedures without an On Error statement. This allows you to add error handling to every procedure that lacks it.

Error Handling settings are customized under Standards. See page 44 for details.

Variable Naming Conventions

The Variable Naming Conventions feature renames existing variables to conform to your naming conventions. Every variable is given a prefix

identifying its type, with additional options for global and module level variables.

If you have public interfaces that should not change, select the “Don’t Rename Public Variables” option and public variables are not renamed. Consider using this option if your project becomes a DLL or library and you don’t want its interface to change.

Variable renaming never changes the name of public variables in class modules. This is because public variables in class modules are essentially the same as class property and method names, and Code Cleanup does not rename procedure and property names.

All variables in your project are analyzed for conversion to the specified naming convention. If a variable is already named with the appropriate naming convention, it is not modified (for instance, variable strTemp is not renamed strStrTemp).

Additionally, if renaming a variable causes a conflict with global variables, the conversion is not applied. For instance, if a global variable intX exists, an integer variable X is not renamed to intX. Variables that are not renamed are listed in the Messages report.

The variable renaming feature cannot apply the correct naming convention if compiler directives are used (e.g. #IF..#END IF) and the same variable name is defined in two different scopes in that case (for instance, one Public and one Private). For situations like this, eliminate the naming convention tags that would cause a conflict under Standards, Naming Conventions, Variable Scope tab. Get rid of the tags that impact your compiler directives.

Naming Conventions are customized under Standards. See page 50 for details.

Remove Line Numbers

Select this option to remove all line numbers from your code. This option is useful for getting rid of out-of-date line numbers, for example if you have added code since adding line numbering. You should not use this option if your code references line numbers via GoTo LineNumber commands. Code Cleanup only removes the line numbers and does not remove any GoTo references to the line numbers, since this will cause your code to behave incorrectly.

When you choose the “Remove Line Numbers” option, you should also select “Cleanup Style” with the option to standardize indentations.

Otherwise, the lines where line numbers are removed may not align properly.

In most cases you won't need this feature of Code Cleanup. We've included this because some developers have used the Code Delivery feature to add line numbers and then want to remove them – especially after editing their numbered code. Of course, we recommend that you always have a backup and do not edit code after it is delivered.

Standards

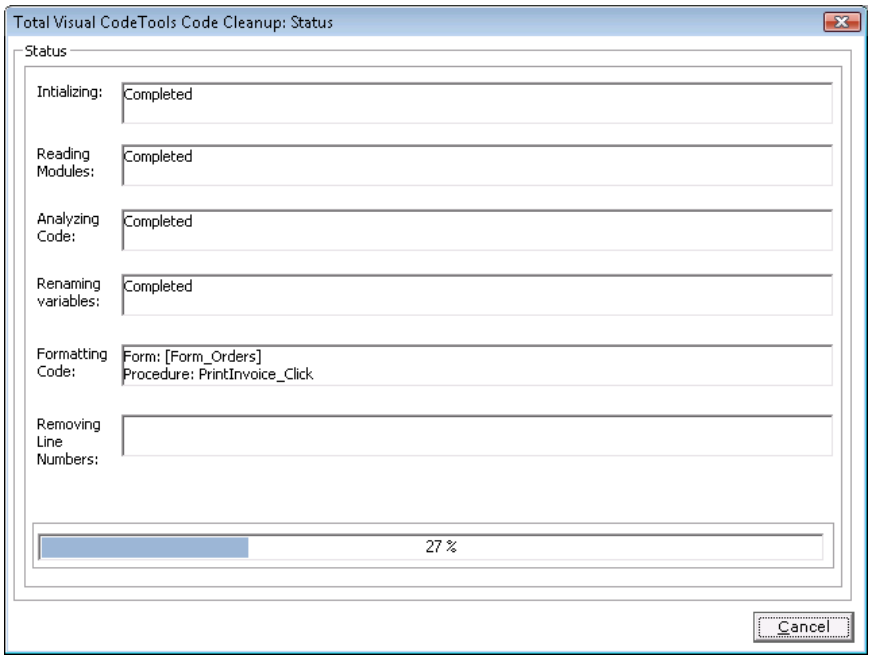
The [Standards] button lets you specify many of the settings that apply to Code Cleanup. Details of the Standards section are provided in **Chapter 3: Managing Standards** on page 23.

Start the Code Analysis

After selecting your objects and options, press [OK] to start processing. Total Visual CodeTools examines the procedures you selected and applies the changes you requested.

Code Cleanup Processing

A screen appears showing the status of the cleanup process. This can take a while depending on the amount and complexity of your code. During this period, your code is only being analyzed and is not modified:

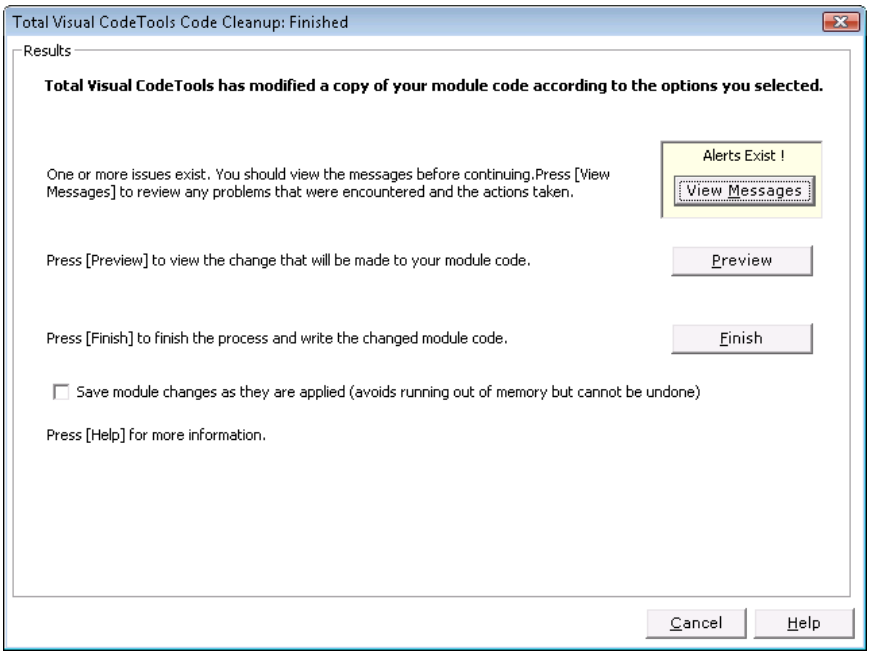


Code Cleanup Status Form

The [Cancel] is available if you want to stop the Cleanup process.

Preparing to Apply Changes

When processing is completed, the following form appears to let you verify the changes you are about to make:



Code Cleanup Finished Form

From here you can:

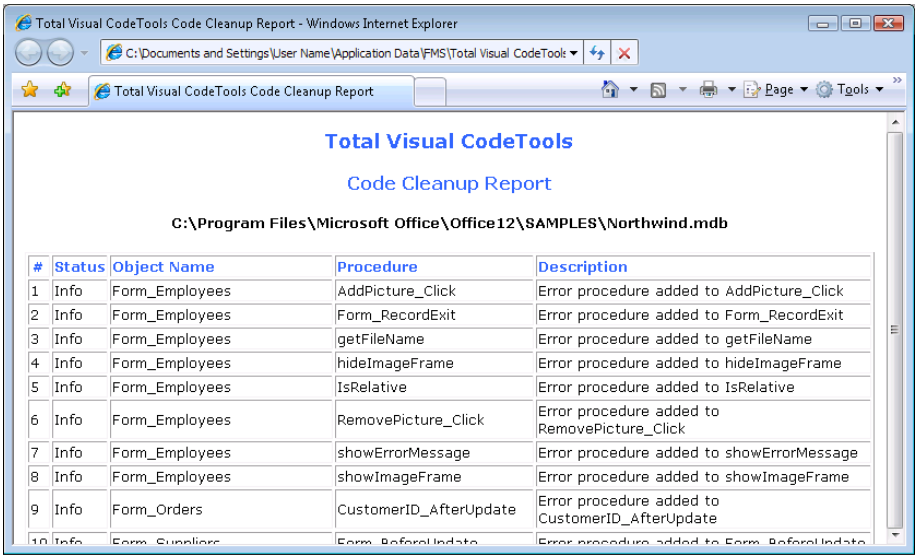
- View Messages about problems encountered and significant changes that were madeor not made.
- Preview the proposed code changes before they are applied to your project, so you can decide whether or not to modify your module code.
- Finish the process by updating your code with the cleaned code.

If errors occurred that left your code in an unusable state or you changed your mind, press [Cancel]. This stops the process without making any changes to your database.

View Messages

If Cleanup encountered errors, a yellow box appears around the [View Messages] button. Be sure to press this button to see a list of problems that Total Visual CodeTools encountered while trying to modify your code.

When you press the [View Messages] button, the messages are presented in an HTML file called Cleanup.htm in your Application Data folder. It is opened in your default Internet browser:



#	Status	Object Name	Procedure	Description
1	Info	Form_Employees	AddPicture_Click	Error procedure added to AddPicture_Click
2	Info	Form_Employees	Form_RecordExit	Error procedure added to Form_RecordExit
3	Info	Form_Employees	getFileName	Error procedure added to getFileName
4	Info	Form_Employees	hideImageFrame	Error procedure added to hideImageFrame
5	Info	Form_Employees	IsRelative	Error procedure added to IsRelative
6	Info	Form_Employees	RemovePicture_Click	Error procedure added to RemovePicture_Click
7	Info	Form_Employees	showErrorMessage	Error procedure added to showErrorMessage
8	Info	Form_Employees	showImageFrame	Error procedure added to showImageFrame
9	Info	Form_Orders	CustomerID_AfterUpdate	Error procedure added to CustomerID_AfterUpdate

View Messages Form

The list is sorted by Status which puts Alerts at the top of the list:

Alerts

Alert items are considered serious issues that you should examine carefully.

- **Option Explicit Added**
  This message lets you know that the “Option Explicit” statement was added to a specific module. Because the addition of Option Explicit can cause compilation problems, you should review this list carefully to determine which modules need immediate attention after running Code Cleanup.
- **EVAL Statement Encountered**
  Cleanup encountered an Eval statement. Code Cleanup cannot evaluate the contents of expressions inside Eval statements.

Because of this, you should examine any code using the Eval statement to determine if it needs to be modified manually.

- **Variable Not Renamed**
  This message occurs when Code Cleanup cannot rename a variable because of a conflict with:
  - A reserved word
  - A public procedure name
  - A private procedure name in the current module
  - A user defined type
  - Another variable (either in the same procedure, or public)
- **User Defined Type or Enum Not Renamed**
  This message occurs when a user defined type or enum is not renamed because of a conflict with another user defined type or enum.
- **User Defined Type Element Not Renamed**
  This message occurs when an element within a user defined type cannot be renamed because of a conflict with another element.

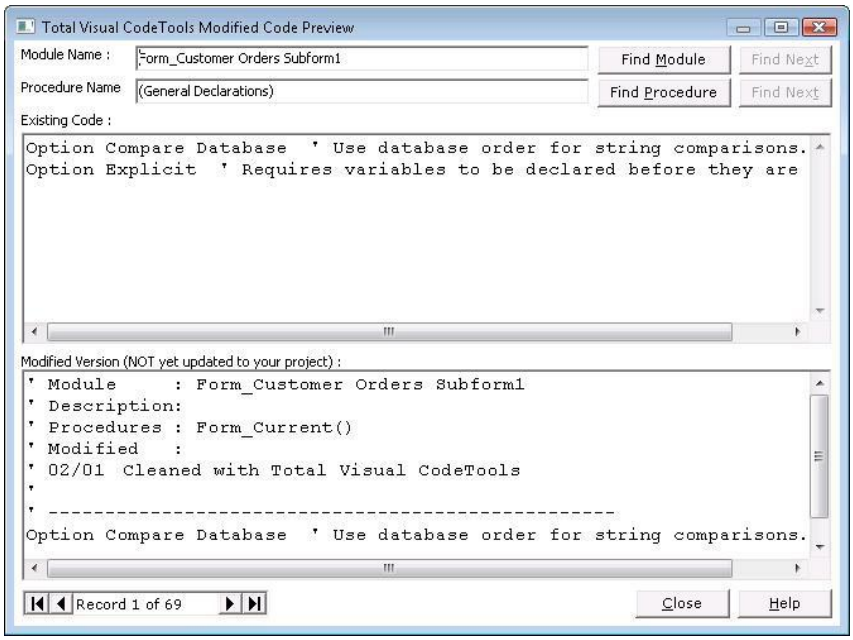
Informational (non-Alert) Messages

Informational items do not indicate serious changes, but are available so that you can better understand what was changed or not changed.

- **Module Comments Not Added**
  This message lets you know that module comments were not added to a specific module because the module already contained comments, and you have the “Always Add” option turned off.
- **Procedure Comments Not Added**
  This message lets you know that procedure comments were not added to a specific procedure because the procedure already contained comments, and you have the “Always Add” option turned off.
- **Error Handler Added**
  This message lets you know that error handling code was added to a specific procedure.

Preview the Code

Press the [Preview] button to preview the code changes. A form appears that shows both the original and modified versions of the code:



Code Cleanup Preview

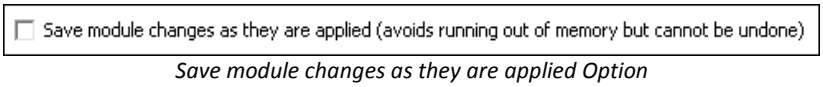
Use the navigation buttons at the bottom of the form to move through the modules and procedures. The Find buttons let you search by module or procedure name.

Apply the Changes

If no unexpected errors have occurred, you are ready to apply the changes. Until you press [Finish], no changes are made—all modified code is stored in Total Visual CodeTools. When you press [Finish], your code is replaced with the modified code.

Save Module Changes as they are Applied

This option is only applicable for Access applications. It does not apply to Visual Basic or other Office programs. For Microsoft Access, Total Visual CodeTools offers an option to save module changes as they are applied:



Save module changes as they are applied Option

If this option is not selected, all the modules including modules behind forms and reports are replaced with the new code and no modules are saved. This allows you to check to see if the changes compile and manually decide to save them.

However, in Access this sometimes causes problems. Because the code behind forms and reports are not saved, the objects remain open on your workspace. This can require a significant amount of memory and cause Access to crash. Additionally, modifications to a parent form or report and its subforms/subreports may fail. In prior versions, the workaround was to select groups of objects separately. This is no longer necessary.

If this option is checked, forms and reports are saved and immediately closed as their code is changed. This reduces the memory requirements of keeping all these objects open, and allows you to select and modify all the forms and reports at one time, while avoiding conflicts among objects. The drawback is the objects are saved before you get a chance to review the changes so they cannot be undone. As always, perform Code Cleanup and Code Delivery on a backup of your database to avoid these problems.

Checking the Results

After applying the changes, ensure that Total Visual CodeTools made the necessary modifications to your code and that your code compiles.

If your database has problems that you cannot easily solve, consider restoring your backup and starting the process again. In almost all cases, problems are the result of code that could not compile correctly before the process was run, or a failure to follow the preparation steps listed on page 126.

Chapter 8: Code Delivery

Code Delivery provides the tools for the final preparation of your code before distribution. Options allow you to line number your code to take advantage of VB/VBA's ability to pinpoint the line number of an error, and to obfuscate your code for situations where you need to distribute your source code, but are concerned about its misuse.

Topics in this Chapter

- **Introduction**
- **Running Code Delivery**
- **Code Delivery Options**
- **Line Numbering**
- **Variable Scrambling**
- **Code Delivery Processing**

Introduction

Code Delivery is used for two primary reasons:

- To simplify application maintenance and bug tracking by adding line numbers to every line of code so an error handler can detect the exact line where a crash occurs (using the VB/VBA function ERL). This information can significantly simplify the process of reproducing and fixing bugs.
- To distribute source code in a form that makes it difficult for the recipient to understand, tinker with, or reuse. By renaming variables to nonsense names, and eliminating comments, blank lines, and indentations, you can easily provide source code that is hard for others to understand. This is particularly useful in VBA hosts where EXEs cannot be created and the source code must be delivered.

Code Delivery should only be performed on a copy of your project and as the final step in preparing it for distribution. You should never edit your delivered code—make changes in your copy.

Preparing to Run Code Delivery

The process of running Code Delivery is very similar to Code Cleanup. Before you run Code Delivery, make sure you follow these steps:

Use a Backup Copy

Code Delivery intentionally modifies your code to make it harder to use by others (especially if you remove comments and obfuscate the variables). Once it does this, you cannot reverse the changes, which means your code could become useless. See page 126 for more information on backups.

Exclusive Access and Compiled State

Ensure project integrity by securing exclusive access, making sure your code compiles, and checking out your objects from SourceSafe. Your code also needs to be updateable. See page 128 for more information.

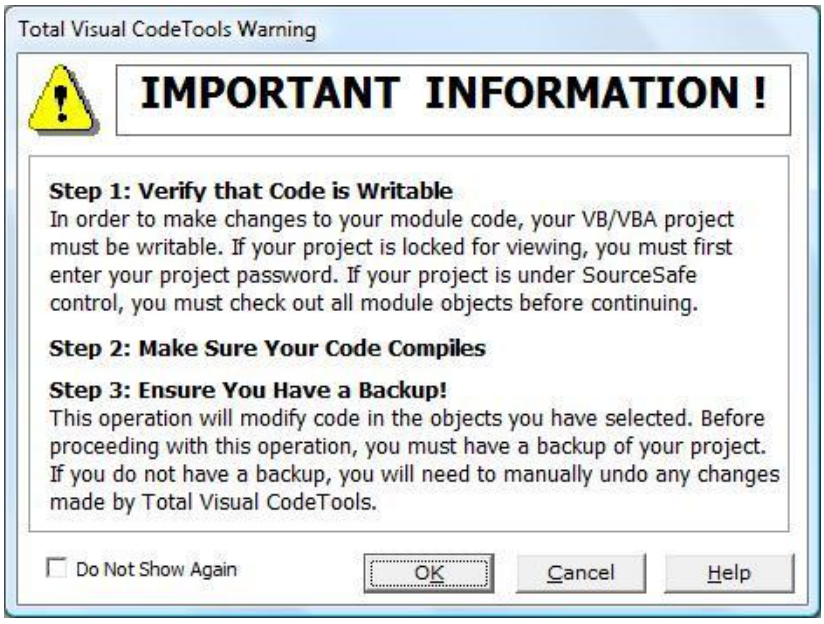
Running Code Delivery

Once the preparatory steps are completed, you are ready to run Code Delivery. Select "Code Delivery" from the Total Visual CodeTools menu or toolbar. There are the steps:

- A warning message to make sure your project is prepared
- Selecting the objects for delivery (current procedure, current module, or some or all modules)
- Specifying the delivery options
- The Code Delivery process runs to present you with the changes to be made
- The ability to preview the changes before applying them (canceling at this point leaves your code unmodified)
- Applying the changes to your project

Warning Message

When you start Code Delivery, a warning message appears:



Startup Warning

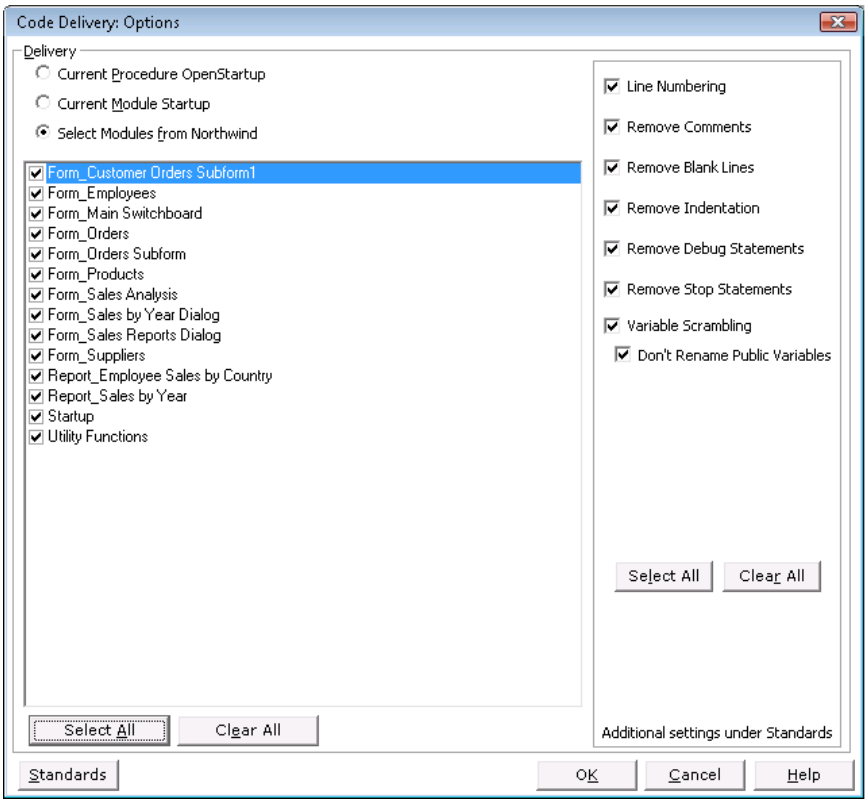
Make sure your code is backed up, compiled, and updateable (checked out of Visual SourceSafe if you're using that). Press [OK] to continue. Otherwise, press [Cancel] and address these issues.

Please remember that Code Delivery cannot recreate your original code after changing it, so a backup is critical.

If you do not want this warning page to appear again, check the Do Not Show Again box. This option is not available if the Standards file has a password, and you'll need to set this on the Standards Delivery page.

Select the Objects

The next screen lets you choose the objects to deliver and the options:



Code Delivery Options

The left side of the form lets you specify the objects to modify. If you have the editor open and the cursor in a procedure, you can apply Code Delivery on just that procedure, just that module, or any or all the modules in your project.

To modify all the code, simply choose the third option and press the [Select All] button.

Code Delivery Options

The right side of the form lets you specify what options to apply to your code. The following options are available:

Line Numbering

Adds line numbers to every line of code so your error handler can use the ERL function to know exactly which line crashed. More details on line numbering are on page 148.

Code Delivery Options

Remove Comments

Remove all comments, except comments that are tagged with the characters specified under Standards **Delivery** on page 58. By default, it ignores comments that start with '!

Remove Blank Lines

Remove all blank lines in procedures and between procedures.

This makes your code more compact and difficult to understand.

Remove Indentations

Remove all indentations and make the code flush left.

Remove Debug Statements

Removes debugging code (e.g. Debug.Print) by placing a comment character in front of Debug statements.

Remove Stop Statements

Removes stop commands by placing a comment character in front of any Stop statements.

Variable Scrambling

Renames all of your variables, constants, user defined types, and enums to nonsense names that someone else would have great difficulty deciphering. Variables are renamed to a letter and a number. The letter is specified under the Standards **Delivery** on page 58. By default, it is V.

Don't Rename Public Variables

Code Delivery provides a Variable Scrambling option to not modify public variables. This prevents renaming public variables and parameters of public methods and properties of classes. This is important if your project is a DLL or library and you don't want its interface to change.

More details on Variable Scrambling are on page 150.

Standards

The [Standards] button lets you specify Code Delivery settings. Details are provided in **Chapter 3: Managing Standards, Delivery** on page 58.

When you've selected your options, press [OK] to start the analysis. Another screen will appear before any of your code is modified.

Line Numbering

Good error trapping should reveal exactly where an error occurs. Ideally, every procedure has an error handler that invokes a global error handler procedure. In that procedure, you can determine the error number, error description, and other information to help you diagnose the problem and exit your application gracefully. Writing the information to a file also improves your ability to diagnose and fix the problem.

If you have procedures without error handling, use the Code Cleanup feature to add your standard error enabler and handler.

ERL Function

VB/VBA has a very useful Erl( ) function to tell you exactly which line caused the error. For more details and examples, see **Add Line Numbers** in the Appendix on page 162

Unfortunately, the ERL function only works if the line which crashed has a line number. Manually adding this is an unbelievably difficult chore.

The Line Numbering feature adds numbers to every line of code in your database, starting with the line number you designate and incrementing by the specified amount. The number is reset for each module.

If a procedure already has line numbering, line numbering is added starting with the highest line number detected plus the increment.

Line Numbering Warning

Line Numbering should be performed as the last step to creating a shipping version of your application. Do not add line numbers to your development project. The Code Cleanup feature can remove line numbers once they are added, however you should not count on this. Removing line numbers may cause problems if your code references line numbers rather than labels.

Existing Line Numbers

By default, line numbers are added to your code from the first line in the first procedure to the last line in the last procedure based on the starting number and increment you specify.

If your code has existing line numbers, line numbers are added a little differently. Basically, the same process occurs, but your existing line numbers are not modified. The lines without numbers are numbered. The numbers assigned are the same numbers that would be assigned regularly (if the procedure had no line numbers) and are larger than the maximum line number already in the procedure.

For instance, if line numbers start at 100 and increment by 10, and the first procedure is numbered 100 through 200, and the second procedure has some line numbers from 300 to 400, the first line numbered in that procedure would be 410.

There may be situations where the same line number occurs in more than one procedure. For instance, if the second procedure has line numbers between 100 and 150, and the first procedure was assigned similar values, there would be two 100, two 110, etc. This is not a problem since labels (and line numbers) are private to each procedure. You just need to make sure your error handler identifies the procedure name in addition to the line number.

Line Numbering Exceeded

If you have the “Line Numbering” option turned on, this message appears when the line number applied by Code Delivery exceeds the maximum limit of 65536. To solve this problem, lower your Starting Line number and Increment values.

Variable Scrambling

The Variable Scrambling feature renames all variables, constants, user defined types, and enums to nonsense names that another user would have great difficulty deciphering. For instance, this function:

```
Function AddCust(Name As String, DoThis As Integer) As Long
  Dim ID As Long, ok As Boolean, exists As Boolean
  customer.Seek "=", Name
  exists = customer.NoMatch
  If exists Then
    ok = (MsgBox("Same name exists, add it?") = vbOK)
  Else
    ok = True
  End If
  If ok Then
    ID = AddCustomerName(Name)
  End If
  Select Case DoThis
    Case 1: customer.Close: code.Close
    Case 2: customer.Close
    Case Else: ErrHandler.Raise
  End Select
  AddCust = ID
End Function
```

is modified to look like this:

```
Function AddCust(V6 As String, V3 As Integer) As Long
  Dim V5 As Long, V7 As Boolean, V4 As Boolean
  V2.Seek "=", V6
  V4 = V2.NoMatch
  If V4 Then
    V7 = (MsgBox("Same name exists, add it?") = vbOK)
  Else
    V7 = True
  End If
  If V7 Then
    V5 = AddCustomerName(V6)
  End If
  Select Case V3
    Case 1: V2.Close: V1.Close
    Case 2: V2.Close
    Case Else: ErrHandler.Raise
  End Select
  AddCust = V5
End Function
```

Variables are renamed based on the Variable Scrambler Root setting under Standards Delivery, plus a number. The number is defined by taking all your variables across all the modules you selected for delivery, sorting them alphabetically and assigning a number starting from one. The result is pretty random.

With the “Remove Indentations” option, the code becomes even more difficult to read:

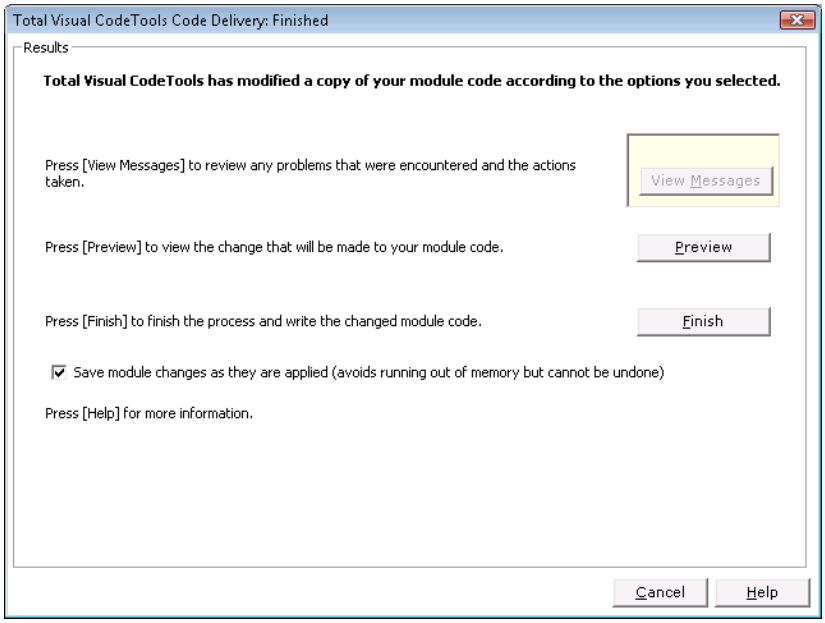
```
Function AddCust(V6 As String, V3 As Integer) As Long
Dim V5 As Long, V7 As Boolean, V4 As Boolean
V2.Seek "=", V6
V4 = V2.NoMatch
If V4 Then
V7 = (MsgBox("Same name exists, add it?") = vbOK)
Else
V7 = True
End If
If V7 Then
V5 = AddCustomerName(V6)
End If
Select Case V3
Case 1: V2.Close: V1.Close
Case 2: V2.Close
Case Else: ErrHandler.Raise
End Select
AddCust = V5
End Function
```

Note that all this does is change the variable names and formatting—your code continues to run the same way it always has.

Code Delivery Processing

Preparing to Apply Changes

When Code Delivery is complete, this screen appears:



Code Delivery Finish Form

Note that none of your code is modified yet. This screen lets you verify the changes you are about to make. From here you can:

- View Messages about problems encountered and significant changes that were made or not made.
- Preview the proposed code changes before they are applied to your project, so you can decide whether or not to modify your module code.
- Finish the process by updating your code with the delivered code.

The options on this form are identical to the Cleanup Finish form, described on page 140.

Chapter 9: Product Support

This chapter provides information on troubleshooting problems that arise and obtaining support for Total Visual CodeTools.

Topics in this Chapter

- **Troubleshooting**
- **Web Site Support**
- **Technical Support Options**
- **Contacting Technical Support**

Troubleshooting

There are many resources available to help you resolve issues you may encounter. Please check the following:

Readme File

Check the README file for the latest product information. The README file is located in the directory where you installed the product.

Product Documentation

We've spent a great deal of care and time to make sure the Total Visual CodeTools manual and help file are very detailed. Check the Table of Contents and Index for your question, and read the appropriate pages.

Web Site Support

The FMS web site contains extensive resources to help you use our products better. Resources include product updates, frequently asked questions (FAQs), newsgroups, information on new versions, betas, and other resources.

Web Site

The FMS web site is located at:

```
www.fmsinc.com
```

News and important announcements are posted here.

Support Site

The main support page is located at:

```
www.fmsinc.com/support
```

From this page, you can quickly locate the other support resources.

Product Updates

FMS takes product quality very seriously. When bugs are reported and we can fix them, we make the updates available on our web site. If you are encountering problems with our product, make sure you are using the latest version.

Product updates can also be checked using the update wizard. See **Using the Update Wizard** on page 15 for details.

Frequently Asked Questions (FAQs)

Common questions and additional information beyond what is in the manual is often available from our FAQs.

Newsgroups

FMS also has general and product specific newsgroups. Connect with FMS Technical Support and other users there. Share your experiences, learn from others, and ask your questions in our virtual community:

`www.fmsinc.com/support/newsgrp.htm`

Or visit our web site for additional information.

Microsoft Patches

Our support site also includes links to Microsoft patches that are related to our products. Make sure you're using the latest versions by checking here or visiting the Microsoft site.

Technical Support Options

FMS is committed to providing professional support for all of our products. We offer free access to our online FAQs and newsgroups. Bug reports, feature requests, suggestions, and general pre-sales questions related to our products are always available at no cost.

Additional maintenance plans are available to provide subscribers with enhanced technical support. This is the best way for you to stay current with the rapidly changing technologies that impact project development, and to ensure you are getting the maximum return from your software investment. Please visit our web site, www.fmsinc.com, for the most up-to-date information.

Features & Benefits	Premium	Incident	Standard
Access to FAQs	✔	✔	✔
Access to Newsgroups	✔	✔	✔
Minor Upgrades/ Bug Fixes	✔	✔	✔
Telephone Support	✔	Per incident	First 30 Days
Email Support	✔	Per incident	First 30 Days
Priority Response Time [1]	✔	✔	
Senior Engineer Support Team	✔	✔	
Email Project for Testing	✔	✔	
Programmatic Code Assistance [2]	✔	✔	
Major Upgrades for Current Version (not between Access versions)	✔	Additional fee	Additional fee
Cost	Annual Fee	Fee Per Incident	Included

1. Response generally within two business days. Actual resolution may take longer depending on complexity of the issue reported.
2. Custom Programming implementation is not provided in our Support Maintenance plans. For products that include a programmatic interface, we can provide instructions for using our programmatic interface, and show examples, but we do not implement this into your projects. This service is available from our Professional Solutions Group.

Premium Subscription

The Premium Subscription is the ideal option for customers seeking the highest level of support from FMS. The annual fee entitles you to telephone and email technical support from a senior support engineer.

From time to time, FMS may release new versions of existing products which add new features. These are point releases (e.g. from version 14.0 to 14.1) and are different from new builds that correct problems in existing features (e.g. from version 14.00.0001 to 14.00.0002).

These point releases are available for a nominal upgrade fee to existing customers. Premium Technical Support subscribers receive these upgrades automatically and for no additional charge during their subscription term.

NOTE: Upgrades between versions (for instance going from Access 2007 to Access 2010) are not considered Point Release Upgrades and are not included in the Premium Subscription.

Subscriptions are available for a twelve month period, and may be purchased at any time. You must be the registered owner of the product to purchase a subscription and the only person contacting FMS for support under the subscription.

Please ensure you have purchased the Subscription you need for Total Visual CodeTools.

Per Incident

Our Per Incident package is available individually or by purchasing multiple incidents in advance. The Per Incident support package provides telephone and email technical support from a Senior Technical Support Engineer for resolving one incident.

An incident is defined as a single question related to one of our products. The Per Incident period is from start to finish (report of the incident to resolution) for a single incident. If you anticipate multiple questions for a single product, we recommend purchasing the Premium Subscription.

Standard Subscription

Our Standard Subscription comes with every product purchased for no additional cost. The standard subscription comes with access to our FAQs and newsgroups, and responses to bug reports and feature requests for that version.

Please note that the person requesting support must also be the registered user of the product. Registration is required and will be requested by our Technical Support professionals.

Contacting Technical Support

If the troubleshooting suggestions and other support resources fail to resolve your problem, please contact our technical support department. We are very interested in making sure you are satisfied with our product.

Registering Your Software

Please register your copy of Total Visual CodeTools at:

```
www.fmsinc.com/support
```

You must be registered to receive technical support. Registration also entitles you to free product updates, notifications, information about upcoming products, and beta invitations. You can even receive free email notification of our latest news.

Contact Us

The best way to contact us is via email at:

```
Support@fmsinc.com
```

Please provide detailed information about the problem that you are encountering. This should include the name and version of the product, your operating system, and the specific problem. If the product generated an error file, please submit that as well.

With email, technical support issues can be more accurately resolved and tracked in our internal technical support system. Email also gives us more time to understand the entire problem and allows our technical support staff to contact the developers with the entire story when necessary. Please bear in mind that a unique issue may involve meetings between the technical support staff and product developers, so your patience is appreciated.

Microsoft Technical Support

FMS only provides technical support for its products. If you have questions regarding Microsoft products, please contact Microsoft technical support.

Appendix: Coding Techniques and Tips

This Appendix provides a resource for writing better Visual Basic and VBA code. This will also help you use the features of Total Visual CodeTools more efficiently, and better understand the Code Cleanup and Code Delivery features, and the code generated by Total Visual CodeTools.

Topics in this Chapter

Writing Better Code

This section provides a variety of Best Practices for writing better Visual Basic/VBA code. By adopting these recommendations, you'll have better skills for writing new code, fixing old code, and managing inherited code. Fortunately, many of these suggestions can be fixed automatically by the Code Cleanup and Code Delivery features in Total Visual CodeTools.

Use Option Explicit

This is not a tip, but a *requirement* for writing good VB/VBA code. If you do not use the Option Explicit statement at the top of every module in your project, you are asking for trouble when you debug your application. By specifying Option Explicit, you tell VB/VBA to force you to declare every variable that you use. This makes it easy to avoid program errors because of misspelled variable names. For example, the following procedure is in a module without the Option Explicit statement:

```
Sub ShowBugs()
  Dim intX As Integer
  Dim intY As Integer
  Dim intZ As Integer

  intX = 1
  imtY = 2      ' Typo here
  intZ = 3

  Debug.Print intX + intY + intZ
End Sub
```

The intent of this code is to assign the values of 1, 2, and 3 to the variables intX, intY, and intZ, and then print the sum of the three (which should be 6). However, this procedure prints the value 4 because of a typo. Instead of setting intY to 2, the code is actually declaring a new variable called imtY (with an "m") and setting it to 2.

Because the module does not contain the Option Explicit statement, the first reference to the new variable declares it automatically, allowing the code to compile properly and hiding the error. If the module contained the Option Explicit statement, the code would generate a compile error, making it easy to track down the mistake during development.

While the bug in this example is relatively simple to spot, the same bug buried in hundreds of lines of code would take far more time to discover.

Using Option Explicit is the most important step you can take to write better VB/VBA code.

Always Add Option Explicit to New Modules

If Option Explicit is not automatically added every time you create a new module, select Tools|Options from your VB/VBA IDE menu, and check the "Require Variable Declaration" option on the Editor tab.

Add Option Explicit to All Modules That Lack It

The Code Cleanup feature of Total Visual CodeTools has an option to **Add Option Explicit** to all modules that lack it. When taking over a project, it's useful to apply this option first. Adding Option Explicit to modules that didn't have it before may trigger compile errors, which will take some time to fix, but is much better than dealing with errors later. See **Option Explicit** on page 132 for more information.

Implement Robust Error Handling

When your VBA application encounters an un-trapped error, you may see a number of interesting and undesirable results. First, a dialog displays the error message, usually with Debug, Cancel, and End buttons. If your application's user is savvy enough to understand the cryptic message displayed, he or she is then unceremoniously dumped into the VBA source code at the line where the error occurred. Such behavior is hardly the hallmark of a professional application. To make matters worse, the source code may be secured, causing an even more cryptic error message to appear. Because of this, error handling is a crucial part of your application development efforts.

Every procedure, no matter how small or insignificant, should have some type of error handling. At the most rudimentary level, there should be an On Error Goto statement that points VB/VBA to a label within the procedure. This label should contain code that, at a minimum, displays a meaningful error message.

Keep the Procedure Call Stack

You may also want to consider creating a procedure *stack*. A stack is simply a list of procedures in the order they are executed. When your application encounters an unanticipated error, you can inspect the stack to see the

order in which the procedures executed. Typically, you would want to use an array to implement a stack. Initialize the array when your application starts. At the beginning of each procedure, place the procedure's name as the last element of the array, moving all other array elements up by one. Just before the procedure exits, it should remove its name from the array.

Although this approach requires extra coding effort, it is well worth it when you try to debug applications, especially when the application and user are at a remote site.

Write Error Log

In addition to gracefully exiting the program, the global error handler that is called by all the procedures should also log the crash environment to a file so you can diagnose the problem. The file should include the error number, error description, error line, the procedure where the error occurred and the call stack so you know how it got there.

Add Error Handling to Procedures That Lack It

The Code Cleanup feature of Total Visual CodeTools has an option to **Add Error Handling** to procedures that lack an On Error command. Based on the error handling structure you specify under Standards, your error handling can use your structure, global error handler, etc. It can also have tokens that are automatically replaced by the module and procedure names, property type, etc. See page 133 for more information.

Add Line Numbers

VB/VBA has an Erl function, which returns the line number where the last error occurred. You can use this function as part of your error handling routines to report exactly where the error occurred. This is extremely important in situations where the code is compiled, and you can't easily step through the program when the crash occurs.

When you know the exact line where a crash occurs, you can often determine the problem with little or no consultation with the end-user. Even if the solution isn't simple, knowing the offending line narrows the tests you need to run to replicate the error.

This technique requires that every line of code be numbered. In this simple example, you can pinpoint which line the random value triggers the crash:

```
Sub SampleErrorWithLineNumbers()
     Dim dblNum As Double
     Dim dblRnd As Double

10   On Error GoTo PROC_ERR
     ' Randomly crashes on a line below:
20   dblRnd = Rnd()
30   Select Case dblRnd
       Case Is < 0.25
40       dblNum = 5 / 0
50     Case Is < 0.5
60       dblNum = 5 / 0
70     Case Else
80      dblNum = 5 / 0
     End Select
90  Exit Sub

PROC_ERR:
100 MsgBox "Value: " & dblRnd & vbCrLf & _
           "Error Line: " & Erl & vbCrLf & _
           "Error: (" & Err.Number & ") " & _
           Err.Description, vbCritical
End Sub
```

Add Line Numbers Before You Ship

Adding line numbers in your code is very cumbersome to do manually. And it's messy to have line numbers in code while you edit it, so you only want to add line numbers right before you ship it. The Code Delivery feature of Total Visual CodeTools has an **Add Line Numbers** option to add line numbers to all your code (see page 148).

Pay Attention to Program Control

VB/VBA provides many constructs for controlling the flow of your program.

More Smaller Procedures is Better than a Large Complex One

For a complex operation that requires many steps, a separate procedure for each step/task is preferable to a single large procedure.

Some developers think new procedures are only needed when it is called more than once. However, breaking up a complex procedure into multiple smaller ones has several advantages even if each is only called once:

- Each procedure has its own set of parameters which makes it easy to see what values it uses.
- Each procedure has its own private variables which should be much fewer than what was in the larger procedure.
- By breaking up the problem into smaller pieces, each procedure can be tested separately, which simplifies debugging.

The rewriting (or refactoring) of existing procedures is complex and risky, so developers often don't do it until the existing procedure is so complex that it's too complicated to support. Better to start with the right structure first.

Use Centralized versus Serial Procedure Calls

A complex operation that requires many steps, each in a separate procedure should be designed to run centrally rather than serially. A serial design has each procedure calling the next one: Procedure A calls Procedure B, which calls Procedure C, etc.:

```
Proc A -> Proc B -> ProcC-> Proc D ->…
```

The problem with the serial approach is that it's very difficult to independently test each procedure due to its dependency on all the procedures before it. It is better to have a "Control" or "Dispatch" procedure to manage the other procedures. It calls procedure A and based on its return value (e.g. True or False to indicate success), proceeds to call Procedure B. Based on the return value of Procedure B, it calls Procedure C, etc. The structure is like this:

```
Function ProcssControl() As Boolean
  Dim fOK As Boolean

  fOK = ProcA()
  If fOK Then
    fOK = ProcB()
  End If
  If fOK Then
    fOK = ProcC()
  End If

  ProcssControl = fOK
End Function
```

This makes it much easier to manage the task flow and test each procedure separately.

Each Procedure Exits at the End

Each procedure should only have one exit point at its end. Avoid using commands such as "Exit Sub" or "Exit Function" within the body of a procedure. Using these commands makes it difficult to understand how the procedure finishes and bypasses any cleanup code that may exist past the exit point. Using multiple exit points can cause bugs that are very hard to detect and debug, since the Exit commands may be triggered by specific conditions that are hard to replicate.

Avoid Gosub Commands

Unfortunately, VB/VBA still has some constructs from the early BASIC days when programs were non-modular, linear lists with required line numbers.

With the exception of On Error commands, these constructs (Goto, Gosub, and Return) have little validity in modern programming. They often lead to messy jumps and logic that are difficult to understand. If you want to write well-structured code, you should never use Gosub…Return statements, but should instead create a procedure.

Declare Variables Correctly

Keep Dim Statements Together

Organize your Dim statements to keep them all together within a procedure, or within the declarations section of a module. By placing all variable declarations at the top of a procedure, you create an inventory of variables belonging to the procedure. You can also perform quality control on your variables by visually scanning the entire list at once without having to hunt through the procedure.

Keep Each Dim Statement on its Own Line

Consider the following code:

```
Dim intX, intY, intZ As Integer
```

A quick glance at the code may lead you to believe that three integer variables are being defined. However, a closer look reveals that intX and intY are not declared as integers but rather as variants. Dimensioning each variable on a separate line makes it easier to identify variables that are not type cast.

Always Declare Each Variable's Data Type Explicitly

If you do not explicitly declare a data type, the variable is a variant, which is far less efficient than a specific data type (see **Avoid Variants** on page 166).

The Code Cleanup feature of Total Visual CodeTools has an option to split multiple variable declarations on one line into separate lines.

This is set on the Cleanup Style page under Standards. See **Split Single-line Declarations** on page 32 for details.

Define Data Types for Constants

VB/VBA lets you specify types for constants using the following syntax:

```
Const strCustomer As String = "Customer"
```

Explicitly Specify the Scope of Module Level Variables

Explicitly specify the scope of a variable as Public or Private in the declarations section of a module or class. By default, variables that are

dimensioned (using the Dim statement) at the declaration level are Private. It is always better, however, to be explicit, since other declarations (such as Enum and Type) are public by default.

Explicitly Type Cast Function Return Values

In addition to declaring variables correctly, the rule holds equally true for functions.

When you create a procedure as a function, you are implicitly telling VBA that the procedure returns a value (if there is no return value, use a sub). If you do not explicitly declare the return type of the function with the **As** keyword, VBA casts the function's return value into a variant. As with variables, this can lead to subtle conversion and logic bugs in your code.

Implicit declaration also prevents the code compilation from detecting incorrect variable assignments. For instance, if you assign a function to a string variable, you get a compile error if the function returns a number. If, on the other hand, you do not specify a data type, your code will compile, but your application may crash when it encounters a value that it cannot handle. Every function should have its return value explicitly set.

Avoid Variants

To make life easier for beginning developers, VB/VBA offers the Variant data type. This data type is flexible because it can represent almost any type of data, and VB/VBA automatically handles all conversions from one type to another. But while this may seem convenient, these strengths also come with weaknesses. Because the variant type is capable of holding any type of data (text, numbers, dates, etc.), it adds storage overhead. Using specific data types for data storage is more efficient than using variants. Also, since VB/VBA automatically converts variants, you are not in control, and you may see unexpected results when conversions take place.

Use Narrow Variable Scoping

All VBA variables and procedures are *scoped* to be available to different portions of the program. For example, a variable declared as Public in a module can be seen or assigned from any procedure in any module in the project. This is the broadest scope. On the other hand, a variable declared within a procedure or passed as a procedure parameter has the narrowest scope—it is only seen by the procedure. Between these two extremes, you also have module level scoping. When deciding on the scope of a variable,

use the narrowest scope possible. If the variable only needs to be used by one procedure, make it private to that procedure. If the variable must be seen by many procedures in the same module, make it a module level variable. Only on rare occasions should you create Public variables. Code that uses narrow scoping throughout is much easier to debug and maintain, and is also much easier to move or copy.

Use "Option Private Module" if you are developing library databases for use by other databases. Add "Option Private Module" to modules that should not be visible to the referencing database.

Convert Data Types Explicitly

In some instances, you may need to convert data from one type to another. VBA makes this deceptively simple by often performing automatic data type conversions for you. This is not necessarily positive; often, these automatic conversions can introduce changes in the data that you do not expect. This is especially true when converting numbers between integers and doubles. Also, the semantics and behavior of automatic data type conversion is often semi-documented or under-documented, and can change between versions of a language. For these reasons, you should use explicit VBA conversion functions, such as CInt, CDbl, CStr, etc. Relying on VBA to perform conversions can lead to bugs that are very difficult to find.

Use Constants to Avoid Hard-Coded Values

Code containing hard-coded values can be difficult to understand and maintain. Consider the following code:

```
Dim dblCost As Double
Dim intQuantity As Integer
dblCost = intQuantity * 1.712
```

This code obviously multiplies a quantity by 1.712 to calculate the cost. But what is 1.712? These types of *magic* numbers offer no information about what is being accomplished, and make the program difficult to decipher. If the number needs to be changed, it may also be very difficult to find.

Next, consider the following code:

```
If Err.Number = 2048 Then
  MsgBox "Error 2048 occurred", 16, "Acme Application"
Else
  MsgBox "Error 2048 didn't occur", 16, "Acmo Application"
End If
```

In this example, hard-coded values cause two problems. Number 16 causes the first problem. What does 16 represent? What type of icon is displayed by the MsgBox function? Second, because “Acme Application” is hard-coded twice, it is easy to introduce typos, such as the one shown in the second MsgBox statement. Finally, by hard-coding literal strings throughout your program, you make it difficult to make global changes later.

You can avoid these problems by replacing hard-coded values with centralized constants or variables, or by placing values in a table. When your application starts, it can read all text values and “magic” numbers from the table into memory, making them readily available to your program. When you remove all hard-coded values, you reduce the potential for errors, and make the program easier to maintain.

Internationalization

Using constants in place of hard-coded values can be crucial if you need to face the issue of internationalization. When you need to translate your application to another language, you will really appreciate code that has all literals and text stored in a central location.

The **Message Box Builder** in Total Visual CodeTools lets you create message box commands visually.

One of its options is to use the built-in VB/VBA Constants rather than hard coded numbers. See page 93 for more details.

Use Variable Naming Conventions

Although the relative merits of different naming conventions can cause heated arguments among developers, the use of naming conventions is generally accepted as good VB/VBA programming practice. Naming conventions add information to variable, object, and procedure names, typically using a prefix or suffix notation to identify the object type. For example, each string variable you create would be prefixed with *str,* identifying those variables as strings. There are many different naming conventions for VBA, split mainly between Visual Basic and Access developers. Beyond code, people also use naming conventions for Access objects (e.g. tables begin with “tbl”, queries “qry”, forms “frm”, command buttons “cmd”, etc.).

The naming convention you choose is not as important as your commitment to use the convention consistently. Naming conventions don’t work well unless you use them throughout your code, so pick a convention and stick to it.

Choose Meaningful Variable Names

It may seem obvious that as a developer you should choose variable and procedure names that convey their purpose. But often in the heat of programming, it is all too easy to resort to cryptic one-letter names like *x* and *y*. For readability, you should avoid such a temptation. Generally, one-letter variable names are reserved for throwaway variables (such as loop counters), and beyond those you should avoid meaningless names.

Unfortunately, meaningful object naming is often at odds with the goal of making code compact. For documentation purposes, you want names that are long enough to adequately describe the variable or procedure, but you don't want names that are so long that they make your code unreadable or difficult to type. Somewhere between the two extremes lies a happy medium: use names that are long enough to convey their purpose, but not so long that code becomes unmanageable.

Add Comments

If you have ever tried to decipher another developer's code (or your own old code), you are no doubt aware of the value of proper commenting. Many times, the purpose of a given piece of code is not readily apparent by reading the code itself. Comments go a long way to providing the information that makes the process of understanding, enhancing, and debugging code much easier.

There are various levels of commenting in VBA coding. These levels closely follow the levels of scoping:

Application (global) comments

Comments at this level explain the flow of the entire application and cover how different components and modules interact. Typically, these comments are kept outside of the application's project database, but can be included in a global module as comment text.

Module level comments

Module level comments explain the purpose and contents of a module. Information typically includes a list of the procedures in the module, their purpose and relationship, and revision comments for the module.

Procedure level comments

Comments for each procedure typically include a brief description of what the procedure does, a definition of its parameters and return value, revision information, and any special situations it requires.

Line level comments

Comments on or above a specific line of code explain the purpose of a variable, or a particular sequence of operations. Typically, it is best to include line level comments on the line immediately preceding the line of code. Comments on the same line as the code itself can be difficult to maintain and make code harder to read.

The **Code Cleanup** feature in Total Visual CodeTools lets you add custom commenting structures to your existing code at the module and procedure levels.
The **New Procedure** and **New Property** Builders automatically adds your comment structures whenever you create a new procedure.

Use Standard Indentation

Control structures should be indented to improve readability and reduce programming errors. Paired structures (such as If...End If, Do..Loop, For..Next, etc.) should use indenting to clearly show where they begin and end. Choose a tab width setting and stick with it. By default, Access uses 4 spaces, but if your code has lots of nesting, 2 spaces may work better. Set your tab width in the VB/VBA IDE's Tools, Options menu.

The **Code Cleanup** feature lets you standardize indentations of procedures, programming loops, and IF blocks on existing code.

See the **Standardize Indenting/Split Colon** option on page 30 for more details.

Indentation also helps you visually group related operations, not just control structures. For example, consider using indentation to group AddNew..Update, Set..Close, and BeginTrans..CommitTrans operations. This makes it easier to see when the code enters and exits these "paired" operations, and to identify if the closing statement is missing.

Avoid Single-Line If Constructs

Placing an If condition and its action on the same line leads to code that is difficult to read and to maintain. There is always the chance that your If statement will need to be modified with an additional line of code, at which point you'll have to break the code out into multiple lines anyway. Also, without a new line for the action part of the If statement, you may be tempted to skip commenting the action altogether. So instead of this:

```
If fInitialized Then MsgBox "I am ready"
```

Do this:

```
If fInitialized Then
  MsgBox "I am ready"
End If
```

The **Code Cleanup** feature lets you convert these single line IF statements into the If..End If block.

See the **Split Single-line IF Statements** option on page 31 for more details.

Use Else with Select Case

The Select Case construct makes it easy for your program to branch based on multiple possible values for a variable or expression. Make sure to use a Case Else clause in your Select Case blocks. Without a Case Else statement, your code does not handle unanticipated values.

For example, assume your application allows the user to add new employee categories, and your application uses VBA code to give employee raises based on their job type. Your Select Case statement to handle raises may look like this:

```
Select Case intEmployeeType
  Case EmpType_Manager
    intRaise = 10
  Case EmpType_Clerical
    intRaise = 5
  Case EmpType_Driver
    intRaise = 2
End Select
```

If the user adds a new “Programmer” category and hires a few Programmers, they are not handled by this code.

If you follow the practice of always adding a Case Else clause, problems like this are easier to handle. For example, the above code could be re-written to prompt the user for the raise amount in the case of new employee types:

```
Select Case intEmployeeType
  Case EmpType_Manager
    intRaise = 10
  Case EmpType_Clerical
    intRaise = 5
  Case EmpType_Driver
    intRaise = 2
  Case Else
    Beep
    intRaise = InputBox ("Enter raise amount")
End Select
```

Use Classes

Class modules are a powerful way to encapsulate properties and related methods. Using classes allows for multiple instancing and simplifies the debugging process, since all related variables and code are in one place. Many VB/VBA programming books describe this subject in detail—if you're not familiar with writing class modules, you should learn.

Avoid Type Declaration Characters

Many of the data types offer the archaic type declaration character. By putting one of these pre-defined characters at the end of a variable or function name, you define the variable's type or the function's return value. For example, the following line of code declares a variable called CName as a string:

```
Dim CName$
```

The dollar sign (`$`) character at the end is a type declaration character that signifies a string variable. Type declaration characters were used in Basic before explicit syntax for type casting, and they still exist today for backward compatibility. Good VB/VBA code, however, avoids type declaration characters, which are considered obsolete and hard to understand. Also, there are only type declaration characters for a small subset of the data types that VBA supports. If you want a variable to be a string, dim it "As String."

Additionally, you should avoid using Def… statements (such as DefInt A-C, which assigns all un-typed variables starting with letters A to C in the module to integers). These constructs are also obsolete, and can lead to code that is difficult to debug and maintain. It also causes problems if you move code from one module to another with different or missing Def… statements.

Conclusion

Writing error-free and maintainable code is hard work—it requires a full time commitment to covering all the details. By following the advice in this chapter, and using the tools available in Total Visual CodeTools, you will be well on your way to writing better and more maintainable code.

For more tips, visit our web site and sign up for our email newsletter where we often offer tips to increase your productivity.

Index

D

E

F

G

I

L

M

N

O

P

R

S

T

U

V

W