

# Project Analyzer 5.1



---

[Introduction](#)

[How does Project Analyzer work?](#)

[Version history](#)      What's new?

[Ordering, prices](#)

[Terminology](#)      What are dead variables anyway?

[Metrics](#)      What are these strange numbers?

[Problem detection](#)

[Comment directives](#)      Customizing the analysis

---

## Menus

### File

[Analyze project...](#)

-

[Generate constants module...](#)

[Archive project files...](#)

-

Exit

### View

Find procedure...

-

[Variables and constants](#)

-

[Metrics](#)

[Call tree](#)

### Report

File details

Procedure details

-

[Module call tree](#)

[Procedure call tree](#)

    All procedures

    Selected file only

    Selected procedure only

Need report...

-

File list...

Procedure list...

Cross-reference report

-

Name shadowing

Library report

-

Problem report

Design quality report

Project summary

## Options

Report to

Display

Printer

File...

Printer setup...

Printer font...

-

General options...

Hypertext options...

Problem options...

## Add-Ins

Save data for Super Project Analyzer...

Super Project Analyzer

-

Project Printer

Project Graph

---

Project Analyzer was written in Visual Basic 6.0 Pro  
by Tuomas Salste, Aivosto Oy, Finland  
© 1994-1999

VBShop - The Visual Basic Shop  
<http://www.aivosto.com/vb.html>  
[vbshop@aivosto.com](mailto:vbshop@aivosto.com)  
Shareware tools for Visual Basic programmers

Project Analyzer 5.0 - Helpfile generated by VB HelpWriter.

# File list

Product Analyzer can produce list of files that belong to your project. To get a list, choose **Report|List files**. A dialog box will appear.

You can get lists sorted by:

- File name
- File type

If you want just a list of file names, leave the **Details** box unchecked. If you check it, the list will contain the following additional data:

- Comments
- Which files need procedures in this file
- Which files this file needs

**Note:** You get the commented lines in the (declarations) section. The program shows those comment lines that come before the first non-comment line, like this:

```
' This is a function module  
' These comments will be reported
```

```
Dim Text As String  
' This comment will not be reported
```

The file list goes to the output device defined with the **Output goes to** radio buttons.

# Procedure list

Product Analyzer can produce list of procedures that belong to your project. To get a list, click the **List procedures** button or choose **Report|List procedures**. A dialog box will appear.

You can select if you want the following data included in the procedure list or not:

- Comments
- Which procedures use this procedure
- Which procedures this procedure uses

You can also choose to list only dead or live procedures.

**Note on comments:** The program will show those commented lines at the beginning of each sub/function that come before the first empty or non-comment line. Commented lines before and after the sub/function declaration line are shown, as well as procedure description (VB 4.0), if available. An example:

```
' This sub was created by N.N.  
SUB MySub (Byval x As Integer)  
Attribute VB_Description=Prints x + 5 on the form  
' This sub does the following:  
' It takes the parameter x and ...  
  
' This line is not shown any more  
Form1.Print x + 5  
  
END SUB
```

The procedure list goes to the output device defined with the **Output goes to** radio buttons.

# Archive Project Files

Project Analyzer helps backing up your source files. It supports three archiver programs, namely ARJ, PKZIP and WinZip. You can also easily build file lists of your project files in order to copy/archive them manually.

Go to **File|Archive project files**. Select the files to be archived in the list box. By default, all source code files belonging to your project are selected. If you want to add files to the list click the Add button.

The **ARJ Command** and **PKZIP Command** fields contain the commands and command switches used to create new archive files.

Sample ARJ Command: [ARJ a -jm -q MYARCH.ARJ !MYPROJ.LST](#)

Sample PKZIP Command: [PKZIP -ex -P MYARCH.ZIP @MYPROJ.LST](#)

Sample WinZip Command: [winzip32.exe -a -P MYARCH.ZIP @MYPROJ.LST](#)

You only have to supply the blue part of the command line; Project Analyzer does the rest. Now you can pack by pressing one of the **Pack** buttons.

By default, a temporary file is used to tell the archiver program what files to pack. If you want to save that list file for later use you can click the **Select** button.

Having named a list file, you can save the list file only if you want. A new button name **List files only** will appear. Click it. The list file will be generated and no archiver program will be run.

# Introduction

Project Analyzer is a utility designed to analyze, optimize and document applications developed with Visual Basic 3.0, 4.0, 5.0 and 6.0.

Project Analyzer makes a two-phase source code analysis. It collects detailed information about all the procedures, controls, variables, classes, etc. in the VB project. It also detects cross-references. You can see the results on the screen or as a report.

**Demo limitations.** The demo is limited to analyzing projects with max 10 files. By ordering the full version you can use the program on projects of any size. There is no known limit how large projects Project Analyzer can handle.

## System requirements

- Windows 95, 98 or NT
- Visual Basic 6.0 installed, or the run-time libraries. Visit <http://www.aivosto.com/vbcatalog.html> to download the needed DLLs.
- Basically, if you can run Visual Basic, you can run Project Analyzer.

**Windows 3.x:** If you are using Windows 3.x, get Project Analyzer version 3.1 from <http://www.aivosto.com/vbcatalog.html>. This is an older version that runs in 16-bit environments.

You only need one version of Project Analyzer for analyzing all your VB 3.0 - 6.0 projects.

## Features

### Browsing of code

- VB Browser, a hypertext code viewer
- Binary extension file viewer (for FRX and other similar files)

### Optimization

- Detection of dead code and other problems
- Metrics like cyclomatic complexity, comment to code ratio, ...
- Name shadowing report
- Call depth report
- Optimization of project groups (Super Project Analyzer)

### Documentation

- Call trees, Project Graph
- Source code printer (Project Printer)
- Comment manuals
- Project summary report
- Library report
- Procedure list with comments
- File list with comments

### Miscellaneous

- Archiving project files with pkzip, WinZip or arj
- Saving and reloading of analysis

See version history for the newest features.

## Why is Project Analyzer useful?

Maintain phase costs rule the software industry. When developing applications, especially those that someone else has to maintain, it's important to keep track on what each sub/function does, and what other procedures they need to work.

This is where Project Analyzer can help you. With it you can produce detailed cross-reference reports with supplied comments. You can view call trees. And you can find dead code, dead variables, and dead constants, and various other problems too. You can get the documents on the screen, on paper, or even on your company Intranet as HTML reports.

If you are interested in more examples on how Project Analyzer can help you to optimize your program, see <http://www.aivosto.com/vbtips/vbtips.html>

## How to use it?

Using Project Analyzer is simple. Open a **.mak** or **.vbp** file with the Open Project command in the File menu and wait. Project Analyzer will collect information about the **.bas**, **.frm**, **.frx**, **.cls**, **.dll** and other VB files in the background.

**Note:** If you are using VB 3.0, remember to save all your files in text format.

The analysis consist of two phases. The first phase collects the most data, and the second one will check for cross-references and find dead code etc. If a menu command or a button is greyed, don't panic, just wait for the analysis to end.

Project Analyzer does **not** write anything into your code, so it's safe to analyze all your projects.

When the analysis has ended, you can save it for faster loading in the future.

[How does Project Analyzer work?](#)

[Terminology](#)

[Ordering](#)

[Back to Contents](#)

# How does Project Analyzer work?

When you start Project Analyzer, or when you choose **Analyze project** from the **File** menu, you get a dialog box asking you to choose a project file (a VB **.mak** or **.vbp** file). By reading that file, Project Analyzer finds out what files belong to the project. After that, Project Analyzer further looks inside these files.

## What kind of files does Project Analyzer analyze?

Project Analyzer examines most file types belonging to your project: .bas, .frm, .frx, .cls, .ctl and other VB source files. As an exception, Project Analyzer doesn't analyze .vb, .ocx, .tlb, .dsr nor .res files. [More about binary files](#).

If you use VB 3.0, remember to save the files in text format. Project Analyzer can't read VB files saved in binary format, other than the .frx files. If the program finds a binary file it will simply ignore its contents. This will result in incomplete analysis, like showing dead variables that aren't dead in reality. (Project Analyzer can analyze DLL files. It does it based on the declarations in your basic code.)

## How does the analysis work?

Project Analyzer reads your files in two phases.

Phase 1/2: Basic information about the structure of your project is collected, including: procedure names, procedure parameter names, variable names, constant names, Type definitions, Enum definitions, control names. Some basic [metrics](#), like lines of code, are calculated too.

Phase 2/2: Cross-references are detected. Project Analyzer scans the whole project to find where procedures are called, variables assigned a value or the values used, constants referenced etc. Some more metrics, like nested conditionals, are calculated too. At the end of Phase 2/2 Project Analyzer calculates additional information based on the cross-reference data: It checks if any procedures, variables, constants, Types or Enums are [dead](#). It also calculates some of the most advanced metrics, like information complexity.

When the analysis is ready, a "Ready!" sign appears at the lower left corner of the program window. Should you want to stop the analysis earlier, you can press the Stop button.

## Can I save the analysis for later use?

Yes you can! Project Analyzer 5.0 re-introduced the [Save analysis](#) feature. The program will save all collected information in a .anl file that you can reload later. This is faster than re-analyzing the whole project from scratch.

## Background processing

Information is collected in the background. So, when Project Analyzer is examining your files you can view the information the program has collected so far. There are some things you can't do when the analysis is in progress. For example, you don't see dead code information before the end of Phase 2/2.

**Remember to use Option Explicit!** Project Analyzer will get better results if you have declared all your variables. This is good programming practice anyway.

[Terminology](#)

[Back to contents](#)





# Report Formats

Project Analyzer has very powerful, formatted report options. You can get the reports

1. **On the screen**
2. **On paper**
3. **In a file for your favourite word processor, WWW browser or Help compiler**

Use the **Options|Report to** menu command or the **Report to** pulldown menu to select the report format. Depending on what you select, you either get a formatted report, or a text-only version.

## 1. To the display

This is the default. The output goes to a special Display window where you can read the text. You have 2 options to take:

- a) A formatted report. The default.
  - b) An unformatted, plain text report. From this report type you can copy parts to the clipboard.
- You can select the type of the report in the **Options|General options** window. [More about Options](#)

## 2. To the printer

If you want, you can choose the output to go to the selected Windows default printer. This report is formatted.

To select a printer, click **Options|Printer setup**. To choose a printer font, click **Options|Printer font**.

## 3. To a file

Project Analyzer can direct its output to:

- a) a Windows text file (plain text)
- d) a [Rich Text Format](#) (RTF) file (formatted report)
- e) an RTF file ready for Help Compiler to turn the report into a Help file (formatted report)
- f) an Internet Hypertext (HTM) file (formatted report)

You can even tell Project Analyzer to run an editor, browser or the Help Compiler to open or process the report after it has been created. To do this, just check the **Run browser/editor after report** check box in the Save report dialog box.

## Running a browser/editor/Help Compiler

Project Analyzer needs to know which program to run after creating a report. By default, the Windows default associations are used. Press the right arrow button in the Save report dialog box to see which program will be launched.

Possible choices for editors/browsers/Help Compiler:

- a) **TXT:** Windows Notepad or any other text editor
- b) **RTF:** Microsoft Word or Windows WordPad, or any other word processor supporting RTF
- c) **RTF for Help Compiler:**
  1. Microsoft Help Compiler (HC.EXE, HC31.EXE, HCP.EXE, comes with VB 3.0 and 4.0, and is in your \VB\HC directory).
  2. Microsoft Help Workshop (HCW.EXE, comes with VB 5.0, and is in the \Tools\Hcw directory of the VB5 CD-ROM).

Use of the Help Compiler is recommended instead of Help Workshop, because Project Analyzer writes a .bat file script for the Help Compiler, which will automatically compile the help file.
- d) **HTML:** Netscape or Explorer, or any other WWW browser

You can change the default by writing your own command line, or by associating the file name extension (.txt, .rtf, or .htm) with your browser/editor. In Win 3.x, you do this from **winfile.exe | File menu | Associate**, and in Win 95, from **Control Panel | View menu | Options | File types**.

### **Multiple reports in one file**

You can add multiple reports in one file by selecting an existing file. If you have selected RTF for Help Compiler, Project Analyzer will also create a Table of Contents page that shows all the reports in that help file.

**Note:** When producing large reports (over 32kB in size) you can get an "Out of string space" error when the output is going to the unformatted display window. The report will be truncated to 32 kB. The solution is to use any other report type that allows reports of any length.

Project Analyzer 5.0 - Helpfile generated by VB HelpWriter.

# Ordering Project Analyzer

Contact information - Future versions

## Contact information

Company	Aivosto Oy Mäkitorpantie 29-31 A 12 00640 HELSINKI FINLAND
Author	Tuomas Salste, a VB programmer since 1993
WWW site	<a href="http://www.aivosto.com">www.aivosto.com</a>
Email	<a href="mailto:vbshop@aivosto.com">vbshop@aivosto.com</a>
Fax	+358 9 728 99690
Languages	English, deutsch, svenska, suomi.

## Future versions

Project Analyzer is constantly under further development. New versions with enhancements can be expected in the future. A licence is valid for one major version number without any need to upgrade. So users of version 5.0 will get version 5.1 for free, for example, but not version 6.0. Special low upgrade prices apply.

Distribution of the free updates is done solely on the Internet, by WWW or FTP. **Getting free upgrade versions requires Internet access.**

[More about pricing](#)

---

## Ordering

Project Analyzer is try-before-you-buy software (also called shareware). The free version of Project Analyzer can be distributed unmodified. You can use it to evaluate Project Analyzer to see how it works. You may evaluate it for as long as you like. You can also give it to your friends.

The free version is limited to projects with at most 10 source code files. To really use Project Analyzer, you'll need to purchase it. That is also called registration. When you purchase, you get keywords for the full version of Project Analyzer with support for projects of unlimited size.

[Prices](#)

## How do I order?

Project Analyzer is available on the Internet only. You will not see it sold in conventional shops. If you do, drop us a note, someone is doing that illegally.

Be sure to read both of the following:

1. [order.hlp](#) for Aivosto's ordering instructions.
2. Special instructions below. They apply especially to Project Analyzer.

## Payment method

The possible payment methods are described in [order.hlp](#). All major credit cards, U.S. checks, and some other methods are accepted.

## **Keyword(s)**

After purchasing, you will get keyword(s) that will enable the full features. Type the correct keyword in the About box when Project Analyzer starts to unlock the program.

The same applies to the add-ins for Project Analyzer. Just type the correct keyword when the add-in starts.

**Note:** There are no physical deliverables. You will not get the program in a box. You will get it via email and WWW/FTP, if you have access to the Internet. In case you don't have an email address, you will receive a letter with the keywords (no diskette).

Project Analyzer 5.0 - Helpfile generated by VB HelpWriter.

# Version History

*This version history lists the major changes.*

*See a more detailed version history in file VERSION.TXT.*

## **Version 5.1 (May 1999):**

- Problem detection made more powerful and configurable

## **Version 5.0 (October 1998):**

- Added Support for Visual Basic 6.0, now supporting VB 3-6
- Saving and reloading of analysis
- VB Browser integrated in the main window
- Call depth report

## **Version 4.2 (August 1998):**

- Need report
- Hotkey conflicts on Problem report
- Support for line continuation character " \_ "

## **Version 4.1 (January 1998):**

- Support for compiler directives (#Const, #If...End If)
- The Hypertext, Procedure Details and File Details windows merged into VB Browser
- New add-in, Project Graph
- Project Printer can do Comment manuals
- Comment directives ('\$ USED) can be used to bypass problem checking

## **Version 4.0 (Autumn 1997):**

- Added support for Visual Basic 5.0
- User defined Types and Enums included in the analysis
- Call tree and Variables and constants window enhanced

# Call tree

You can see the procedure and file dependencies in the form of a call tree. The tree is available in three formats:

## 1. Call tree in a separate window

Use the **View|Call tree** menu command to see an expandable view of the dependencies. The call tree types in this window are:

**1.1.** Module calls. This tree shows those other modules that are needed for a module to run. A module may use procedures, variables, constants, and also Type and Enum declarations coded in other modules. It may also call another Form's controls and properties.

**1.2.** Procedure calls. This tree shows the procedure calls inside your project.

### The buttons

<b>Expand</b>	Expands the call tree from the cursor onwards.
<b>Browse</b>	Shows the selected piece of code in <u>VB Browser</u> .
<b>Report</b>	Turns the tree into a textual report.

*Max call depth* at the bottom of the window shows how deep the call tree has been at a maximum. This number will increase as you expand the tree.

If a procedure calls itself or another procedure that then eventually calls the first procedure back, it will be marked **<recursive call>** in the tree.

## 2. Textual report

The reports are available using the **Report|Procedure call tree** and **Report|Module call tree** menu commands. These call trees are the same as on the Call tree window. Three subtypes of a procedure call tree report are available:

**2.1.** All procedures

**2.2.** Procedures in selected file only

**2.3.** The selected procedure only

## 3. Project Graph

The Project Graph add-in shows graphical call trees.

**Note 1:** The call trees are available after analysis phase 2 has completed.

**Note 2:** You can see dependencies in hypertext or list form too.

**Note 3:** The program may hang if your call trees are extra large. In this case, see smaller call trees at a time. It is recommended not to take the All procedures call tree on anything but small projects.

# Variables and constants window

Project Analyzer can show a list of all global and module-level variables/constants either as a textual report or on the screen. To see the screen list, either press the **Vars & consts** button in the main window, or select the **View|Variable and constant list** menu command.

## Options

- Constants
- Variables
- Global vars/consts
- Module-level vars/consts
- Live definitions (those that are used somewhere in the program)
- Dead definitions (those that are not needed for your program). [More info](#)

## Buttons

Declaration	Opens <a href="#">VB Browser</a> with the (declarations) section where the the selected variable/constant is defined, highlighting it
References	Shows a list of all references and assignments to the selected variable/constant. If it's dead, you the reference list is empty. You can also create a reference report in that window.
Report	Creates a report of all variables/constants that are currently on the list.

**Note 1:** Variables and constants information is available when analysis phase 1 is complete. References and "dead" or "live" status are available when phase 2 is complete. To refresh the window, close and reopen it after the analysis has completed.

**Note 2:** To see procedure-level variables or constants, use [VB Browser](#).



# Problem report

This function is a bit outdated. See [Problems](#) for a newer one.

-----

The Problem report lists some minor but sometimes annoying things. It is recommended that you take the report only after the analysis has been fully completed.

- 1. Basic files saved as binary.** Save as text to work with Project Analyzer **and** for better reliability with VB. This problem exists only in Visual Basic 3 (and 2).
- 2. Files without Option Explicit.** It is good programming practice to always declare your variables.
- 3. Dead code.** By removing dead code, you can optimize the size and speed of your program.
- 4. Variables and functions without a type.** If you don't give a type with the "Dim x As Type" statement, you get a Variant type. Variants take more memory than other data types, thus slowing your program down. It is professional programming practice to use a specific data type whenever possible. Using Option Explicit will help you to spot even more of these implicit Variants.
- 5. Forms missing an icon** even if an icon would be needed.
- 6. Sizable forms without Resize event.** Put a Resize event to these forms, or set their BorderStyle to something that is not resizable.
- 7. Forms with HelpContextID = 0.** If you use context-sensitive help in your application, Project Analyzer warns you of possible contextless forms. This warning only applies to projects with a help file.
- 8. Hotkey conflicts.** This section warns if some menus or controls on your form share a hotkey that the user can select with the Alt+key combination.
- 9. Miscellaneous.**
  - EXE file without title and icon

## Terminology

Project Analyzer 5.0 - Helpfile generated by VB HelpWriter.

# Metrics

To monitor their programming efforts, software engineers often use some simple metric, like lines of code or EXE size. These are the most basic metrics, although they are not very sophisticated.

You can still use lines of code or kB as size estimates, but Project Analyzer knows more. It can tell you about the understandability, complexity, and reusability of your code.

You can see metrics in many different ways

- The **View|Metrics** window lists procedures and modules sorted by a given metric.
- Project summary report summarizes the overall performance.
- The Design quality report also shows the overall performance.
- The problem detection feature warns if some metric exceeds its limits.
- VB Browser shows metrics for each module and procedure.

Below you can read a brief discussion on understandability, complexity and reliability, and the various metrics used to estimate them.

## Understandability

Bad understandability will most probably result in more errors and maintaining problems. You can estimate the understandability of your program with the following metrics:

**Lines of code / procedure** (View|Metrics, Design quality report)

Procedures longer than 50 lines are often too long. An alarm limit could be 100 lines.

**Comment to code ratio** and **whitespace to code ratio** (Design quality report)

The more comments in your code, the easier it is to read - and understand. Whitespace is also important for legibility.

**Length of identifiers** (Design quality report)

The longer your variable, procedure etc. names are, the more likely they are to be descriptive.

**Cyclomatic complexity** (View|Metrics, Design quality report)

Target for less than 10. See below.

**Depth of conditional nesting** (see View|Metrics)

Target for less than 5. See below.

## Complexity

There are many kinds of software complexity:

- Structural complexity relates comes from conditionals and loops, that is, the flowgraph of the program.
- Psychological complexity means how difficult it is to understand a program. This is very much related to structural complexity.
- Informational complexity is about how much data comes into a procedure and how much goes out.
- Mathematical/computational complexity is about how much time and memory it takes to execute an algorithm.

High complexity may result in bad understandability and more errors. Complex procedures also need more time to develop and test.

The best measures for estimating the complexity of a procedure with Project Analyzer are:

- Lines of code
- Cyclomatic complexity
- Informational complexity

- Structural fan-out

These measures deal mostly with structural and informational complexity. See below for more information.

## Reusability

Reusability is a magic word in programming. Measures for reusability are

- Reuse ratios reported by Super Project Analyzer (reuse in a group of projects)
- Structural fan-in. If it's high, the procedure/module is called (reused!) many times.
- Number of class modules

## What do the different metrics mean?

**Note:** You can see most of the metrics only after the whole analysis has been completed.

### Lines of code

As simple as it may seem, lines of code is quite a good measure of how complex a program is. Project Analyzer calculates lines of code as follows:

$$\text{Lines of code} = \text{Total lines} - \text{Commented lines} - \text{Empty lines}$$

Lines of code does not include control declarations in a .frm file.

### Cyclomatic complexity

(McCabe)

Cyclomatic complexity is a measure of the structural complexity of a procedure. The higher the number, the more complex the procedure, and the harder it is to maintain it. Cyclomatic complexity for VB procedures is calculated as follows:

$$\text{Cyclomatic complexity} = \text{Number of Branches} + 1$$

Branches are caused by IF, SELECT CASE, DO...LOOP and WHILE...WEND statements.

"Normal" values for cyclomatic complexity range from 1 (very simple) to 9 (moderately complex). If cyclomatic complexity is more than 10, you may want to split the procedure. If it's over 20 you can consider it alarming.

Cyclomatic complexity is the minimum number of test cases you must have to execute every statement in your procedure. This is important information for testing. Carefully test procedures with the highest cyclomatic complexity values.

An average cyclomatic complexity, as well as the distribution of complexity, is reported on the Design quality report.

**Note:** Cyclomatic complexity often gives quite high values for procedures with long SELECT CASE statements.

### Nested conditionals

*Nested conditionals* metric is related to cyclomatic complexity. Whereas cyclomatic complexity deals with the absolute number of branches, nested conditionals is only interested in how deeply nested these branches are.

If you have a procedure with deeply nested conditionals, you should consider splitting it up. Those procedures can be very hard to understand, and they are quite error-prone too.

### Nested loops

*Nested loops* is a very rough estimate of the mathematical complexity of a procedure. The more nested loops

there are in a procedure, the more likely it is that those loops take up a significant amount of time to execute.

## Structural fan-in/fan-out

(Constantine & Yourdon)

### For procedures:

**Structural fan-in = the number of procedures that use this procedure**

**Structural fan-out = the number of procedures this procedure calls**

### For modules:

**Structural fan-in = the number of modules that use variables, constants, or procedures in this module**

**Structural fan-out = the number of modules whose variables, constants, or procedures this module needs**

A high structural fan-in denotes reusable code.

The higher the structural fan-out, the more dependent that procedure is on other procedures, and more complex too.

## Informational fan-in/fan-out and informational complexity

(Henry & Kafura)

Lines of code, cyclomatic complexity, or structural fan-out are not perfect in predicting the "real" complexity of a procedure. For example, a procedure may access a number of global variables and be very complex without having to call many other procedures.

*Informational fan-in* = Procedures called + parameters referenced + global variables referenced  
Informational fan-in estimates the information a procedure reads

*Informational fan-out* = Procedures that call this procedure + [ByRef] parameters assigned to + global variables assigned to  
Informational fan-out estimates the information a procedure returns

Combined, these give a new metric: *informational fan-in x fan-out*. This is reportedly good in predicting the effort needed for implementing a procedure, but not so good in predicting complexity. To predict complexity, we need a new metric: informational complexity. It is calculated as follows:

**Informational complexity = lines of code x (informational fan-in x informational fan-out)**

## What are the target values?

There are no widely accepted target values for most of the metrics. You will have to make up your own goals. Take the Design quality report for some of your projects and decide the good and bad values yourself.

## Literature

See your library for software engineering related books. They usually have a chapter or two about metrics too. Some books you could check:

*Tom Manns and Michael Coleman: Software Quality Assurance.*

*Carlo Ghezzi, Mehdi Jaszayeri, Dino Mandrioli: Fundamentals of Software Engineering*

*Roger S. Pressman: Software Engineering*

If you are really interested in structural complexity measures, there is a book that makes a thorough mathematical examination of 98 proposed measures for structural intra-modular complexity. This is for the advanced reader only.

*Horst Zuse (1991) Software Complexity. Measures and Methods. Walter de Gruyter. Berlin - New York.*

Project Analyzer 5.0 - Helpfile generated by VB HelpWriter.

# Rich Text Format

(version 2.0 onwards)

Project Analyzer can produce formatted reports and save them in a Rich Text Format (RTF) file. A number of popular word processors read RTF files, including Microsoft Word and Windows 95 WordPad. Project Analyzer also produces RTF for the Help Compiler to turn reports into **.hlp** files.

[More information on report formats](#)

## MS Word

A special option for Microsoft Word users is the **Create Microsoft Word style RTF** check box on the **Save Report** dialog box. This option may produce ugly reports for other word processors (for example, for WordPad).

If this option is set, Project Analyzer optimizes the RTF reports for Word. It uses styles like Normal, Heading 1, Heading 2, ... You can easily create a Table of Contents for your report with Word.

## RTF for Help Compiler

You can also create an RTF file for the Microsoft Help Compiler. This way, you can turn the reports into Help files. Project Analyzer will create the necessary .prj file and launch Help Compiler automatically.

The **Create Microsoft Word style RTF** checkbox has no effect on RTF for Help Compiler.

# Options

## On double click show

These options let you specify what happens when you double click the **file list** or the **procedure list** on the main screen.

- 1) **Report** produces the standard reports for a file or a procedure
- 2) **Detail window** shows file details or procedure details in a separate window
- 3) **Hypertext window** opens the hypertext window with the selected procedure or (declarations)

## Show reports to Display as

- 1) **Formatted text** Default. Reports to Display are formatted.
- 2) **Unformatted text**. Plain text reports in a text box. Choose this if you want to select part of the report for Copy & Paste.

[More about report formats](#)

## Refer to files by

- 1) **File name**. Use the file name (without path) to refer to any files in your project.
- 2) **Module name**. If a file has a module name, use it to refer to the file. Otherwise, use the file name.

## Treat variables that are assigned to but not used

- 1) **as live variables**
- 2) **as dead variables (default)**

By default, Project Analyzer thinks that if a variable is assigned to but its value is not used anywhere, the variable is a dead one. These variables are included in the [Problem report](#) and overstricken in VB Browser. If you don't like this, change the setting to option 1.

## Use Windows default colors

By default, Project Analyzer uses bright background colors in several windows. Check this option if you prefer ordinary gray backgrounds (in fact, the color used is vbButtonFace, that is, the Windows default button face color).

Hypertext options

# Generate constants module

(VB 3.0 projects only)

The use of the CONSTANT.TXT and DATACONS.TXT files in a project tends to take up a large amount of memory, because these files include so many **global const** declarations, and only a few of them are needed in one project. Including a large number of dead constants in a project will fill up the global symbol table, as well as decrease the performance of the program.

If you use Project Analyzer, you can solve the problem easily. When you are developing your application, you can include the CONSTANT.TXT and DATACONS.TXT files in your project; you don't need to cut and paste any declarations to another global module. When your project is about to be ready, just analyze it with Project Analyzer, and select **File|Generate constants module**. This command will generate a new module including only those constant declarations that are needed by your application.

After generating a new module, remove the old one (CONSTANT or DATACONS) from your project and add the new one. You're done.

**Note 1:** This command applies mainly to VB 3.0 projects. VB 4.0 doesn't require you to declare a vast amount of constants, but has predefined constants instead.

**Note 2:** This command is available only after analysis phase 2 has completed.

Project Analyzer 5.0 - Helpfile generated by VB HelpWriter.



# Library report

This report lists all DLL, OCX and VBX files used by the project.

All declared DLL procedures are listed, and if they are not used, they are marked as "dead".

VBX and OCX files included in your project are listed at the end of the report, as well as any libraries that you have checked in the References dialog box (VB 4.0 and later). If your project file (.vbp) includes information on the version of the library, that is shown too.

Project Analyzer 5.0 - Helpfile generated by VB HelpWriter.

# Super Project Analyzer

Super Project Analyzer is an add-in tool for Project Analyzer. It is purchased separately. It also needs the full version of Project Analyzer to work.

A Super Project is a group of ordinary Visual Basic projects that share some files. Super Project Analyzer analyzes these project groups and reports which procedures, variables and constants etc. are shared or dead.

## Instructions

To use Super Project Analyzer, save a couple of normal .vbp/.mak analyses using the **Tools|Save data for Super Project Analyzer** menu command. This will write a .sud (Super Project Data) file for each VB project. This file includes information on the modules, procedures, variables and constants used by that project.

After saving a few (at least 2) .sud files, start Super Project Analyzer (**Tools|Super Project Analyzer**). Then open the .sud files with the **Project|Add project** command in Super Project Analyzer. You can now see the modules, procedures, variables and constants used by all of these projects.

**Note:** In the free version of Project Analyzer, Super Project Analyzer can only handle projects with less than 10 source files. If you have the full version of Project Analyzer, Super Project Analyzer can handle up to 25 source files per project without a separate purchase. The full Super Project Analyzer, of course, has no such limitations.

## Features

Select **Shared only** to limit the display to those elements that are included in at least 2 projects. **Shared Only** may also show dead elements that are included in at least 2 project but are not used by them.

Select **Dead only** to see which elements are not used by any project. You may consider removing them.

**Note:** Super Project Analyzer does not list VBX, OCX, or FRX files. It does list DLLs and DLL procedures.

## Reports

To get a report on what you see on the screen, press the **Report** button, or select the **Report|Report what you see** menu command.

Select the **Report|Reuse report** menu command to get a report on reuse ratios. The report includes reuse ratios for modules, procedures and variables & constants by project and for the whole super project as well.

**Reuse ratio for modules = Shared modules / All modules in project**

**Reuse ratio for procedures = Shared procedures / All procedures used in project**  
(thus excluding dead ones)

**Reuse ratio for vars&consts = Shared vars&consts / All vars&consts references in project**  
(thus excluding ones that are completely dead or that are only assigned to)

## Saving a Super Project definition

To save a Super Project definition to a .sup file, use the **File|Save Super Project** menu command. You

can later open this file to skip opening each project separately.

**Note:** In addition to the .sup file, you will need to have the saved .sud files in order for the analysis to work. The .sup file only includes the file names of those .sud files.

## **Ordering**

To use Super Project Analyzer, you will need to order both Project Analyzer and Super Project Analyzer.

Prices

[More about ordering](#)

Project Analyzer 5.0 - Helpfile generated by VB HelpWriter.

# Project Printer

Project Printer is an add-in tool for Project Analyzer. To use it, you will need to purchase both Project Analyzer and Project Printer.

Project Printer is designed to document VB source code. It can

- Report your source code on the printer or any other output type
- Document references
- Generate hyperlinked source documents in a HTML or HLP file
- Generate a comment manual

Project Printer is not just a plain ordinary code printer. It can

- highlight variables, constants, Types, Enums and procedure calls
- overstrike dead variables, constants, Types and Enums
- list cross-reference information
- report various metrics and problems along with the code
- generate Table of Contents
- generate hyperlinked source code documents (like the VB Browser)
- take just your source code comments and make a manual

You can start Project Printer by selecting the **Tools|Project Printer** menu command.

**Note 1:** Project Printer gives best results after the analysis has been fully completed.

**Note 2:** The free version of Project Printer can only handle projects with up to 10 source files. The full Project Printer has no such limitations.

## Output types

Project Printer is versatile. In step 2/3, it asks you to select the output type. Do this in the main window in the ordinary way, as would do when printing any other report.

**Paper.** Ordinary formatted source code printout with Table of Contents. Optionally with metrics and 'called by' list.

**Help file.** The easy way to browse a project. Project Printer generates a help file that is very much similar to the Hypertext window of Project Analyzer. Calls are hyperlinked. The Help file is always there for you to take a quick look at your project, even on computers without Project Analyzer.

**HTML file** for Intranet documentation. Calls are hyperlinked. Useful for a group of programmers. Even store your code bank in HTML.

**RTF file.** Format the report in any way you want with your favourite word processor.

**Other types.** You can get the report in any other report type Project Analyzer supports, for example, on the screen, but the results are not as usable as in the above types.

## Report types

Project Printer can do 2 report types:

- 1) Source code
- 2) Comment manual. The Comment manual option reports comments only, formatted using a special syntax. More information

## Options

Project Printer has a number of options in step 3/3. Default values are set based upon which report type has been selected when you continue from step 2/3. It's normally wise to use the default values, but sometimes you'll want to change them.

**Source style.** **Simple** is a fast, plain text report. **Enhanced** parses through the code, highlights identifiers and overstrikes any dead ones. In HTML and HLP, it also sets hyperlinks for references.

**Show contents.** On paper, generates a Table of Contents with page numbers at the end of the report. In a Help file, it's on a separate page. In other report types, it is at the start of the report.

**List global/module-level vars and consts.** Check this if you want a list of vars/consts declared in (declarations). The list includes information on where a variable or constant has been referenced or assigned to, or if it's dead.

**Show incoming calls.** Lists the procedures that call this procedure.

**Show outgoing calls.** Lists the procedures that this procedure calls.

**Show metrics.** Shows various programming metrics, like complexity etc.

**Show problems.** Shows various programming problems as selected in **Options|Problem options**.

**i-30976Group Subs/Functions/Events/Properties.** Normally, the procedures are reported in the order they appear in the source code. This option groups them by procedure type.

## Ordering

To use Project Printer, you will need to order both Project Analyzer and Project Printer.

[Prices](#)

[More about ordering](#)

# Prices for Project Analyzer 5.1

All prices below are in U.S. dollars. You can pay in your own currency, but please email [vbshop@aivosto.com](mailto:vbshop@aivosto.com) for the price. The prices are subject to change.

## Project Analyzer

Licence	Single user	Additional users	Site licence
Project Analyzer 5.x	\$90	\$90 / user	\$900
Upgrade from Project Analyzer 4.x to 5.x	\$40	\$40 / user	\$200
Upgrade from Project Analyzer 3.1 to 5.x	\$50	\$50 / user	N/A

The price includes all Project Analyzer versions with major version number 5. You can expect a major version number change after the next major Visual Basic version has come out. Minor version upgrades can be expected several times a year. There will be a special upgrade price for licenced users when new major versions come out.

An upgrade licence is valid if you have obtained an earlier version of Project Analyzer that the upgrade is for. You can upgrade to Project Analyzer 5.x from versions 3.1, 4.0, 4.1 and 4.2.

A *single user licence* is valid for one user only. If more than one user uses Project Analyzer, you will need to purchase a *multiuser licence*. If lots of programmers in your organization need Project Analyzer, you can consider a site licence. A site licence covers a single organization for an area of up to one hundred miles (160 km) in radius. The price for a site licence price is equal to the price of a multiuser licence for 10 users.

If you already have a a single or multiuser licence and want to upgrade for more users, you will get a discount for the price you have already paid. Email [vbshop@aivosto.com](mailto:vbshop@aivosto.com) for details.

Add-ins	Single user	Additional users	Site licence
<u>Super Project Analyzer</u>	\$30	\$30 / user	\$300
<u>Project Printer</u>	\$50	\$50 / user	\$500
<u>Project Graph</u>	\$50	\$50 / user	\$500

The above add-ins are optional. They need the full version of Project Analyzer to work.

An add-in licence is valid for all future Project Analyzer versions (the possible versions 6, 7 etc). If you have bought add-ins for versions 3.1 - 4.2, there is no need to upgrade: your licence is still valid in version 5.x.

## How do I pay?

The possible payment methods are described in [order.hlp](#). All major credit cards, U.S. checks, cash, and some other methods are accepted.

## Delivery

There are **no deliverables**. You will not get the program physically in a box. You will get it via [email and WWW/FTP](#), if you have access to the Internet. In case you don't have an email address, you will receive a letter with the keywords.

[More about purchasing](#)



# Terminology

Procedure	A Sub or Function, either a VB procedure or a DLL declaration.
Dead procedure	A sub/function that is not executed at run-time. It can be 1) a procedure that is not called anywhere in the program 2) a procedure that is only called by other dead procedures
Live procedure	The opposite of a dead procedure. The following kinds of procedures are always considered as live procedures.  1) Event Subs. There is no way to detect at design-time if an event occurs at run-time.  2) Procedures that implement an interface. Procedures in the abstract base class are analyzed normally, but in the child class, procedures like <code>MyBaseClass_MyMethod</code> are considered live, because there is no way to detect if they are actually called at run-time or not.  3) Public members of public classes. Public classes are often called from outside the project they are in, so Project Analyzer can't analyze the calls to them. See Public class below.  4) Procedures defined as used by <u>comment directives</u>
Dead constant	A constant whose value is not used in the code.
Dead variable	<i>1) For non-object variables (e.g. Integers, Variants, user-defined Types...):</i> A variable whose value is not used in the code. - General rule: by default, a variable that is assigned to but whose value is not used, is defined as a dead variable. You can override this in <u>General options</u> . - Exception: a ByRef procedure parameter that is assigned to is never dead, because its value may be used in the procedure where the call came from.  <i>2) For object variables (e.g. Class, Control, Form variables):</i> An object variable that is not assigned nor referenced to. - An object that is referenced is never dead. - An object variable declared as <code>Dim x As New ObjType</code> is never dead. - An object variable declared as <code>Dim x As ObjType</code> is not dead if it is assigned to, for example, by <code>Set x = y</code> .
Dead Type/Enum	A user-defined Type is dead if it doesn't appear in the code. An Enum is dead if neither its name nor any of its members appear in the code.  Public Types and Enums in Public classes are never considered as dead, because they may be used by other projects. (See Public class below)
Public class	A Public class can be one of the following: a) .cls files with <code>Instancing &lt;&gt; Private</code> (ActiveX EXE, DLL or OCX projects) b) .ctl files with <code>Public = True</code> (OCX projects) c) .dob files (ActiveX EXE or ActiveX DLL projects)
Variable assignment	A variable is given a new value with one of the following statements <code>x = 1</code> <code>Let x = 1</code>



Set x = y  
For x = 1 To 100  
For Each x In group

Variable reference      Every occurrence of a variable that is not an assignment.

Assignments and references can exist in a single statement, like below:  
MyObject.MyProperty = 3  
This is a reference to MyObject and an assignment to MyProperty.

Object variable      A variable that can hold an object. The object can be, for example, a Class, a Form, or a Control. Variants can also hold objects, but they are classified as non-object variables, because there is no way at design-time to detect if a Variant holds an object at run-time.

Global      An entity that is available throughout the whole project. Example: a variable declared Global or Public in the (declarations) section.

Module-level      An entity that is available inside one module only. Usually declared Private in the (declarations) section.

Procedure-level      An entity that is available inside one procedure only. Examples: procedure parameters and variables declared inside a procedure.

You can use [Comment directives](#) to tweak the analysis. For example, you can make the analysis ignore certain dead code.

[Metrics described](#)  
[Contents](#)

# VB Browser

The main window of Project Analyzer is called VB Browser. VB Browser has three pages:

1. Hypertext
2. Module details
3. Procedure details

VB Browser works after analysis phase 1/2 has completed, although it gives more results after phase 2/2.

## 1) Hypertext

The hypertext page is a read-only code window with sophisticated cross-reference features. You can see called procedures and declarations of constants, variables, Types and Enums by clicking underlined words.

By right-clicking a highlighted word, you get a menu with cross-reference information. This works with all global and module-level definitions, but not procedure-level ones.

### Color guide

These are the default colors for things in the Hypertext window.

<b>Procedure</b>	Call to a Sub, Function or Property
<b>DLL procedure</b>	Call to a DLL Sub or Function
<b>Variable</b>	Reference or assignment to a variable or constant declared in (declarations)
<b>Local variable</b>	Reference or assignment to a procedure-level variable or constant
<b>Module</b>	Reference to a module name
<b>Comment</b>	Comment
<b>Rejected code</b>	Code not included in your project because of #If...#End If compiler directives
<b>Dead thing</b>	<u>Dead</u> procedure, variable, constant, type or enum (after analysis phase 2/2 is ready)

### Options

You can change the colors of different types of code by selecting **Options|Hypertext options**. You can also enable or disable the **overstriking** of dead things.

---

**Hint:** **Project Printer** can save your source code in hypertext format into a Windows Help file. The result is quite similar to the Hypertext window, but you no longer need Project Analyzer to view the file. You may, for example, distribute your code in a Help file with hyperlinks.

## 2) Module details

The Module details page shows information on single files, either normal VB code files or DLL files too.

### Basic files

- Size in kilobytes
- File version: An internal version number of the file. This version is not necessarily the same as the version of your VB. Not all file types have a version number.
- Controls (Form files only)
- List of files that use this file (after phase 2/2)
- List of files that this file uses (after phase 2/2)
- Total lines, and lines of comment and white space
- Subs and functions: Name, type (functions only), lines of code, and 'Dead' status if the procedure is not used by your project. Double click a procedure to jump to it.

- Variables, constants, Types and Enums defined in this file: Global and module-level definitions.
- Controls on Form, UserControl and PropertyPage files

### **DLL files (DLL, EXE, ...)**

DLL's are analyzed like normal Basic files, except:

- No variables, constants, procedures, Types or Enums in the DLL are detected
- Files that are called by this DLL are not analyzed
- The lines of code in a DLL cannot be analyzed
- DLL file version is not analyzed

Only those procedures that were declared with a Declare statement are shown.

## **3) Procedure details**

The Procedure details page shows information about a specified sub, function, or property procedure. It also shows information about (declarations) sections and declared DLL procedures.

The following information is shown:

- Procedure declaration
- The 'deadness' of the procedure (after phase 2)
- Total lines and lines of code (doesn't apply to DLL procedures)
- Various metrics (after phase 2)
- Procedures that call this procedure
- Procedures that are called by this procedure (doesn't apply to library procedures)

Some parts of the information will not be available before the analysis phase 2 has been completed. 'Wait...' indicates that the analysis is not complete yet.

# Binary Files

Project Analyzer cannot analyze most binary files. Some of the file types that Project Analyzer doesn't analyze are VBX, OCX, TLB, DSR and RES. Moreover, Project Analyzer cannot analyze VB files saved as Binary in VB 3.0.

However, Project Analyzer can analyze DLL files based on Declare statements. It can also analyze FRX files, as well as CTX, PGX, DOX and DSX files which are similar to FRX files in structure. The rest of this topic is about the FRX view that you can display by double-clicking a FRX file (or similar) in the main window of Project Analyzer.

## FRX/CTX/PGX/DOX/DSX files

Project Analyzer can examine the contents of a binary extension file that is there if you saved your Form file in text format (VB 3.0 has this option, later versions do it automatically) and the form contained pictures or any other binary data. In Visual Basic 4.0 and earlier, there are only FRX files available. In VB 5.0 and later, there are lots of other files that are essentially of the same type, for example, a CTL file is often accompanied by a CTX file.

These files include the binary properties of your controls. You can see what's inside a binary extension file by double-clicking it in the main window of Project Analyzer, or by selecting File Details.

Project Analyzer cannot show all kinds of binary data, just the most usual types of them. Some supported properties are:

- PictureBox.Picture
- Form.Icon
- Form.MouseIcon
- TextBox.Text (if it's a multiline box)
- ListBox.List and ListBox.ItemData

If Project Analyzer cannot show a picture, it will try to show the data as text.

You can extract a picture from your VB project by saving it to a file. Just click the **Save As** button.

Project Analyzer 5.0 - Helpfile generated by VB HelpWriter.

# Project Graph

Project Graph is an add-in tool for Project Analyzer. To use it, you will need to purchase both Project Analyzer and Project Graph.

Project Graph builds graphical call trees. It can build them at both module and procedure level. The call trees Project Graph builds are more detailed than those given by the [Call Tree](#) window, because Project Graph shows both

- 1) Forward calls, originating from the selected module or procedure, and
- 2) Backward calls, coming from other parts of your project to to the selected module or procedure

Double-click a node to move in the tree. Right-click a node to display a context-sensitive menu.

**Expanding.** Because call trees easily grow very large, the initial depth of a call tree is limited to 3 levels. However, you can manually expand branches by right-clicking a node where the tree can be further expanded (indicated by an arrow like this: -> 4) and selecting **Expand**. You can also move to a node and press the **x** key to call expand. Sometimes expanding will produce a spaghetti-like call tree. To avoid this, expand nodes in up-to-down order.

**Printing.** You can print the graph at any time as it appears on the screen. Project Graph will fit the graph at the center of the paper.

**Note 1.** Project Graph works after the analysis has been fully completed.

**Note 2.** The free version of Project Graph can only handle projects with up to 10 source files. The full Project Graph has no such limitations.

## Ordering

To use Project Graph, you will need to purchase both Project Analyzer and Project Graph.

[Prices](#)

[More about ordering](#)

# Comment manual

Comment manual is a feature of the Project Printer add-in. It creates a manual-like report based on comments in your source code.

Comment manual takes all comments immediately before and after your Sub/Function/Property declaration line and makes a report out of them. All comments before the Sub/Function/Property line are taken, as well as all comments after it up to an empty line or a normal code line. This latter rule holds for comments at the start of the (declarations) section too.

Normal comments are reported as such, just comment marks (') and Rem statements are removed. There are 2 syntax rules that you can use to format the output, namely Titles and Parameter tables.

## 1. Titles

```
' MyTitle: Text line 1
' Text line 2
```

This will produce a title line with text "MyTitle", and below that "Text line 1" and "Text line 2". The following would also give the same result:

```
' MyTitle:
' Text line 1
' Text line 2
```

## 2. Parameter tables

```
[Param] Descr
```

This will create a little table with 2 columns and default headings, like this:

Parameter	Description
Param	Descr

A parameter table ends when there are no more commented lines with brackets.

## 3. General tables

```
[*3*] [Heading1] [Heading2] [Heading3]
[a] [b] c
[d] [e] [f]
```

[\*n\*] marks the start of a general table with n columns of equal width. The [Headings] after [\*n\*] create bolded column headings for the table. You can create a table without headings by putting [\*n\*] alone on a line. A table may contain a maximum of 5 columns. A table ends when there are no more commented lines with brackets.

Note: You may omit the brackets [ ] around the last column if the row has more than one column.

This is what will be created:

Heading1	Heading2	Heading3
a	b	c
d	e	f

#### 4. Ignoring a line

~ (tilde) at the start of a commented line prevents the line from being included in the manual.

#### Example

This is an example of a commented procedure:

```
Function PrintText(x As Integer, y As Integer, Text As String) As Boolean
' Prints text on the screen at (x, y)
' [x] The x-coordinate in pixels
' [y] The y-coordinate in pixels
' [Text] The string that will be displayed
' ~Don't show this line
' Returns:
' True if successful
' False in case an error occurs
' Remarks: Calls Err.Raise 12345 if Text is too long
```

(and the real code would start here)

This is what you would get:

#### **PrintText**

##### **Syntax**

Function **PrintText**(x As Integer, y As Integer, Text As String) As Boolean

Prints text on the screen at (x, y)

<b>Parameter</b>	<b>Description</b>
x	The x-coordinate in pixels
y	The y-coordinate in pixels
Text	The string that will be displayed

##### **Returns**

True if successful  
False in case an error occurs

##### **Remarks**

Calls Err.Raise 12345 if Text is too long

#### Project Printer

# Comment directives

Comment directives are a way to tell Project Analyzer to behave in non-standard ways. You can put them into your code to ignore dead code, for example.

## Comment directive syntax

```
'$ <directive> [ : <directive> [ : <directive> ...]]
```

Comment directives always start with '\$

The prefix \$ marks the start of a comment directive line. Rem \$ is not accepted.

You can put multiple directives on one line by separating them with a colon (:).

## List of possible directives

1. USED
2. END

### USED

Marks things as being in use (and not "dead").

Syntax:

```
USED [<option>] [BEGIN]
```

<option> is optional, and it can be one of the following:

PARAM	Directive affects procedure parameters only
PROC	Directive affects procedures only (both VB and DLL procedures)

If you omit <option>, the directive affects all things, namely procedures, variables, constants, Types and Enums.

The optional parameter BEGIN marks the start of a BEGIN...END block (see Scope below for more information).

Examples:

```
'$ USED  
Sub MyProc ( x )
```

Defines MyProc and x as used

```
'$ USED PARAM  
Sub MyProc ( x )
```

Defines x as used, but doesn't affect MyProc.

```
'$ USED PROC  
Sub MyProc ( x )
```

Defines MyProc as used, but doesn't affect x.



## **END**

Ends a BEGIN . . .END directive block. See Scope below for more information.

## **Scope**

What code do the directives affect? There are 3 alternative formats you can use.

### 1) Next line

```
'$ USED  
Dim MyVariable As Integer
```

A directive on an otherwise empty line affects the immediately following line. In this case, MyVariable is marked as used.

### 2) Current line

```
Dim MyVariable As Integer '$ USED
```

A directive after a code line affects the current line. In this case, MyVariable is marked as used.

### 3) Begin...End block

```
'$ USED BEGIN  
Dim MyVariable As Integer  
Dim Text As String  
'$ END
```

The directive affects all code before the next '\$ END directive. In this case, both MyVariable and Text are marked as used.

You can nest multiple BEGIN . . .END blocks.

[Terminology](#)

[Contents](#)

# Saving and reloading of analysis

The **File|Save analysis** menu command lets you save the results of the analysis in an **.anl** file. Reloading an **.anl** file is much faster than re-analyzing the project from scratch.

You can save the analysis once it has ended. Saving after phase 2/2 will store the complete analysis, saving after phase 1/2 will only store the first phase.

To reload an analysis, open the **.anl** file instead of the **.vbp** or **.mak** file.

**Note:** A single **.anl** file is not enough to reload the analysis. You will have to have the actual source code files on your system too. This is because the **.anl** file doesn't include all the information, like the complete source code.

**Reloading modified projects.** Project Analyzer will reload the unmodified files of your project from the **.anl** file, and re-analyze those files that are affected by modifications. If you make many modifications, the reloading will be slower because many files will have to be re-analyzed.

Example: A project has 3 files: A, B, and C. These files call each other like this: A calls B, and B calls C.

File	Modified?	Phase 1	Phase 2
A	no	reloaded	reloaded
B	no	reloaded	re-analyzed
C	yes	re-analyzed	re-analyzed

File B has to be re-analyzed in phase 2/2, because it depends on file C in a way that could have changed because of the modifications in C.

The format of **.anl** files may be different between various Project Analyzer versions. If your old **.anl** file is not compliant with the current Project Analyzer version, just re-analyze the whole project.

# Problem detection

Version 5.1 onwards

Project Analyzer detects several types of programming problems. The problems are of four categories:

1. Optimization. These problems affect the speed and size of the resulting program negatively.
2. Style. These problems are related to the programming style. They don't necessarily lead to problems in the short run, but they often lead to worse understandability and errors.
3. Metrics is a sub-category of Style. You can set target values for different metrics and monitor if some part of your program exceeds the limits. [Read more about metrics.](#)
4. Functionality. These problems affect the run-time functionality of the program.

The problems are further classified by severity.

✖ Severe problem. These are problems that you should fix before releasing your program.

⚠ Warning. These are problems that should be fixed but that are not critical.

i Information. These are the least severe problems. You may think some of them are not real problems at all.

## How do I see the problems?

Check the bottom of the main window for a problem list. Press the report button for a report. Additionally, [Project Printer](#) can list problems at places where they occur.

## Configuration

You can choose which problems to show and which ones to hide. You can create quick check filters and strict filters for giving your code a final polish. The predefined filters are:

Dead code	Check deadness only
Disable checking	Disable all problem checking
Functionality	Report problems affecting functionality only
Optimizations	Report problems related to optimization only
Strict	Report all problems
Style	Report problems related to style issues only

Select a filter and click Add to create a new filter based on the selected one.

## Problems listed

### 1. Optimization

Problem	Description
Dead procedure/declaration - " - (called by dead only)	A procedure (or a DLL procedure) is not used A procedure is effectively dead, but it is still called by other dead procedures.
Dead variable/constant	A variable or constant is not used
Variable assigned only	A variable is never read
Dead type/enum	A user-defined type or Enum is not used
Unused file	A basic module is not used <a href="#">Read more about the terminology</a>

Function without type specification	A function does not have a return type defined. By default, the type is Variant. A Variant needs more memory than other types.
Variable without type specification	A variable does not have a type defined. By default, the type is Variant. A Variant needs more memory than other types.
Function with type character	A function is declared ending in one of the old-style \$#%!&@. Instead of MyFunction\$( ), you should use MyFunction( ) As String, etc.
Variable/parameter with type character	A variable or procedure parameter is declared ending in one of the old-style \$#%!&@. Instead of Year%, you should use Year As Integer, etc.
Procedure without code	A procedure exists without any code in it.

## 2. Style

Option Explicit missing	A module is missing its Option Explicit statement. You should always use Option Explicit. Using it makes VB require declaration of all variables. This way you avoid using unnecessary implicit Variants. Even better, you get rid of some nasty errors caused by typing errors and variables with similar names.
Default control naming	A control has not been named with a descriptive name. VB has generated a default name instead, like Text1, List2 etc.
Short name	A name is too short and not likely to be descriptive. The name can belong to a control, procedure, variable, constant, Type or Enum - just about anything. You can set the limit yourself, and Project Analyzer reports names that are equal to or below the limit.
Goto/Gosub statement found	Use of Goto/Gosub is bad programming practice and should be avoided when possible.
Exit statement found	Use of Exit means an unstructured routine. Check if you can build a similar control structure without the Exit statement.
Uncommented code	A procedure has no comments. It may be hard to understand.
Excess scope	A part of your program is accessible from other modules. However, other modules don't use it. Check if you should declare the part as Private instead of Public. It is good programming practice to use as tight a scope as possible to prevent other parts of the program calling and modifying parts that they shouldn't have anything to do with.
Scope declaration missing	A part of your program does not have a defined scope. Instead, it is using a default scope as given by VB. VB's default rules are somewhat complicated and may lead to excess scope or a scope that is too limiting. Determine if it should be Private or Public and declare the scope accordingly.
Global found, use Public	You use the old keyword Global. Nowadays, use of Public is recommended. Simply replace Global with Public.
ByVal/ByRef missing	A parameter has no ByVal or ByRef declaration. The default is ByRef. ByRef can lead to unwanted side effects causing variable values to change unexpectedly.

Basic module saved as binary

Generic type Object used

Make all parameters explicitly ByVal or ByRef, and use ByVal whenever possible. ByRef is good programming practice with object variables, but in other cases you should avoid it. - VB 3.0 does not have the ByRef keyword.

Project Analyzer can't analyze the file because it was saved as binary. Problem with VB 3.0 only.

A variable is declared As Object. While this is sometimes useful and completely acceptable Visual Basic, it isn't always necessary. If you know what kind of object the variable will contain, you should declare it as that type, As Listbox, for example. This way VB will check the code better at design-time and even give you warnings if you accidentally happen to call members that don't exist.

### 3. Metrics

Informational fan-in x fan-out exceeds limit

A metric exceeds its limit. [Read more about metrics](#). You can set the limits yourself.

Informational complexity exceeds limit

Cyclomatic complexity exceeds limit

Loops nested too deeply

Conditional statements nested too deeply

Number of code lines exceeds limit

### 4. Functionality

End/Stop statement found

Your program ends here. Check if you really want to end your program like this. It may be completely as you planned, though, and if you think everything is OK, leave it as is.

Error handler missing

A procedure has no error handler. Your program may crash if a run-time error occurs in this procedure.

Delayed error handler

A procedure has no error handler, but postpones all error handling by On Error Resume Next. Errors are not handled in this procedure, but may be handled in the procedure(s) that call this procedure. This may be completely as you planned, or you may have forgotten to add an error handler.

Form missing Icon

A Form doesn't have an icon that is required. A default icon, generated by VB, will be shown.

Resizable Form missing Form\_Resize

The Form\_Resize event is missing from a Form that users can resize at run-time. Your application may look odd if you don't respond to Resize events.

Form missing HelpContextID

Your program uses context-sensitive help (F1), but a Form was found that has no HelpContextID set.

Hotkey conflict

Two or more controls or menu items share a hotkey. For example, there are options &Save and &Search in the same menu.

Click event missing

A CommandButton does not do anything when clicked.

Timer event missing

A Timer does not fire an event at defined intervals.



