



This article shows 12 advanced code analyses with [Project Analyzer](#), a professional code analyzer for the Visual Basic programming language. These tips are written for experienced Project Analyzer users looking for new ways to analyze their VB source code.

Save the forests! If you print this document, print 2 pages per sheet, on both sides of the paper. Or just read on the screen.

1. List parameters of a certain data type

Sometimes you might want to get a listing of all procedure parameters of a certain data type. Say you want to know all Byte parameters, all parameters accepting MyClass or all ParamArrays.

1. Select *View/Variables, Consts and parameters* or just press *Shift+V* to display the *Variables, constants and parameters* window.
2. Uncheck *Consts* and *Variables*, keep *Params*.
3. In the *Filter by* combo, select *Type*.
4. In the text field to the right, write the data type (Byte or any other data type).
5. Press *Enter*.

This trick also lets you find all ParamArrays or Optional parameters. Instead of *Type*, select *Declaration*, and write ParamArray or Optional in the text field.

2. Find constants with the same value but different name

This feature lets you spot unintentional duplicates. You might have constants that are exactly for the same purpose, yet their names differ. Suppose your code defines MAXUSERS=100 and MAX_USR=100. When you need decide to change MAXUSERS, you probably need to change MAX_USR as well, otherwise the code might fail. To fix this potential error, remove MAX_USR and use just MAX_USERS, if they indeed are the same thing.

1. Press *Shift+C* to open the *Constants and Enums* window.
2. Select *Same value, different name* in the combo at the top.

Naturally, declaring different constants with the same value is completely OK if the purpose is different, such as MAXUSERS=100 and MAXFILES=100.

In this window you can also find the opposite: *Same name, different value*. These are problem cases, since the values differ. Your code might be using the wrong value.

By the way, VB6 defines two versions of vbCancel:

```
VBA.VbMsgBoxResult.vbCancel = 2  
VBRUN.DragConstants.vbCancel = 0
```

Which one are you using? If you just write vbCancel, this defaults to 2. This is good for using with MsgBox. However, your drag & drop code might fail if you just wrote vbCancel and not the fully qualified form, either DragConstants.vbCancel or VBRUN.DragConstants.vbCancel!

3. Document a large Enum

Suppose you have a big and important Enum..End Enum definition containing 200 constants but no documentation. You wish to check out which values the individual constants have and if they are all used.

1. Bring up the Enum in the hypertext view.
2. Right-click the Enum name and select *Report*.

You will get a listing such as this one:

| Enum EFileDialogFlags in PicMain | Value | Reads |
|--|--------------|--------------|
| OFN_ALLOWMULTISELECT = &H200 | 512 | 0 |
| OFN_CREATEPROMPT = &H2000 | 8192 | 0 |
| OFN_FILEMUSTEXIST = &H1000 | 4096 | 1 |
| OFN_HIDEREADONLY = &H4 | 4 | 2 |
| OFN_NOCHANGEDIR = &H8 | 8 | 0 |
| OFN_NODEREFERENCCELINKS = &H100000 | 1048576 | 0 |
| OFN_NONETWORKBUTTON = &H20000 | 131072 | 0 |
| OFN_NOREADONLYRETURN = &H8000& | 32768 | 1 |
| OFN_NOTESTFILECREATE = &H10000 | 65536 | 0 |
| OFN_OVERWRITEPROMPT = &H2 | 2 | 1 |
| OFN_PATHMUSTEXIST = &H800 | 2048 | 2 |
| OFN_READONLY = &H1 | 1 | 0 |
| OFN_SHAREAWARE = &H4000 | 16384 | 0 |
| OFN_OPENDEFAULTS = OFN_FILEMUSTEXIST Or OFN_PATHMUSTEXIST Or OFN_HIDEREADONLY | 6148 | 1 |
| OFN_SAVEDDEFAULTS = OFN_OVERWRITEPROMPT Or OFN_PATHMUSTEXIST Or OFN_HIDEREADONLY Or OFN_NOREADONLYRETURN | 34822 | 0 |

In the Value column you can see the actual numeric value of each constant. In the Reads column you get the number of times the constant is being used by your program. A zero value indicates an unused constant. Check out to see if you could remove it or if it's worth keeping for future use.

4. Get a quick overview of a new program

Got a new legacy project to work on? Not sure about its structure? The *Form.Show diagram* gives a good idea of what a program does and how it is structured. It shows the order in which the forms show each other.

1. Press *Ctrl+F7* to run *Enterprise Diagrams*.
2. Select *Form.Show order* in the combo box in the top-left corner.
3. Press *Ctrl* and *+* to add all forms to the diagram
4. Press *View*.

This diagram requires the Enterprise Edition. You can get a bit similar diagram in the Pro Edition via *Project Graph* (press *F7*).

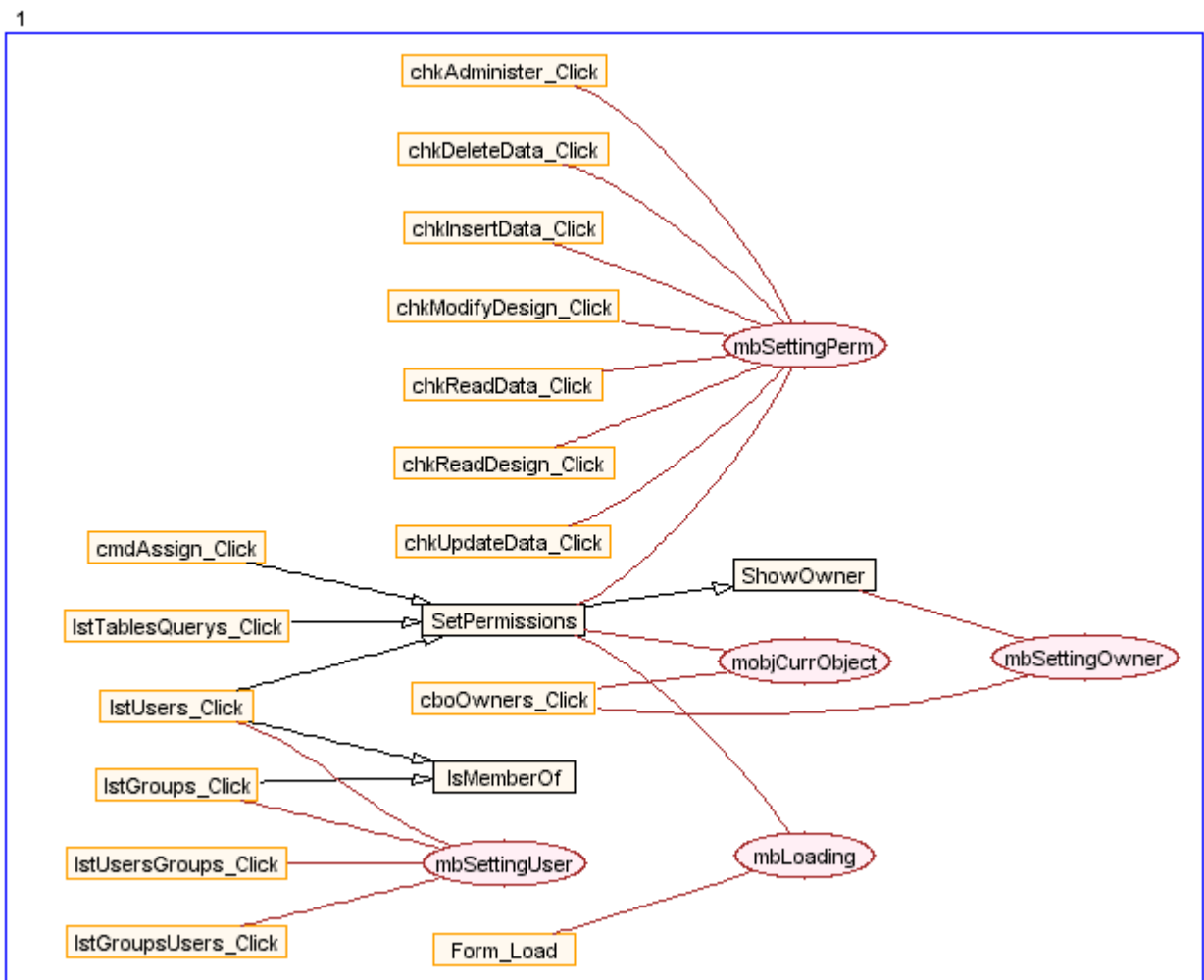
5. See how a class works inside

To learn how the methods of a class work together and which class-level variables they use, get a *Class cohesion diagram*. This diagram shows method calls and variable usage within a single class, module or form. No external calls are displayed, which keeps the diagram compact.

1. Press *Ctrl+F7* to run *Enterprise Diagrams*.
2. Select *Cohesion* in the combo box in the top-left corner.
3. Double-click the class or module you wish to diagram.
4. Press *View*.

There's a sample below. It's actually from a form, not a regular class. A form is basically just a special type of a class, so it serves as an example.

You can see how variable `mbSettingPerm` is shared by many procedures. `SetPermissions` is a kind of a central procedure on this form. You can make a guess of what the variables are used for even if you haven't read the code.



Note: This diagram requires the Enterprise Edition.

The intended use for a Class cohesion diagram is to find unrelated groups of methods and variables within a class, which can then be split into a number of smaller classes. The idea is that each cohesive class consists of just one group of related methods and variables. If there are several unrelated groups (having no variable or method in common), the class should be split. Nothing prevents you from using the diagram to just document some classes, their methods and variables, even if you don't intend to split the class.

6. List API callback functions

When you make use of advanced API calls, you sometimes need to provide a callback function. You use the syntax `AddressOf procedurename` to pass a function pointer in a DLL call. Project Analyzer lets you list these calls.

1. Press *Shift+R* or select *View/References* to open the *References* window.
2. In the *Type* combo box, select *Address*.
3. In the *Source file* and *Target file* combo boxes, select *<All>*.
4. Press *Search* to get all `AddressOf` references.

The resulting list includes all lines with the keyword `AddressOf` in them. As far as API callbacks are concerned, this trick works best for classic VB, where the only use for `AddressOf` is indeed with API callbacks. In VB.NET you can use `AddressOf` for other purposes as well, so the result list can contain other things in addition to callbacks.

7. Run multi-project analyses via .lst files

If you ever need to analyze 50 or 100 individual projects, you will get sick of adding them manually via Project Analyzer's *Add project(s)* button.

Don't worry, there's a solution. Create a `.lst` file with Notepad. In this file, list each project, project group or solution file name you need to analyze like this:

```
c:\projects\proj1.vbp  
c:\projects\proj2.vbp  
c:\projects\proj3.vbg
```

Then just open the `.lst` file in Project Analyzer. This trick requires the Enterprise Edition.

8. Publish a code snippet on your web site

When you wish to publish a syntax-formatted code snippet on your web site or intranet, Project Analyzer can give you the HTML.

1. Bring up the code you wish to publish in the hypertext view.
2. Select the lines you wish to publish.
3. Right-click and select *Copy as HTML*.
4. Paste the raw HTML to your HTML editor.

If you use a more advanced web page editor (Word, maybe) that accepts formatted input instead of raw HTML, select *Copy* instead of *Copy as HTML*.

9. Hide comments

Someone has really spoilt your code with all those comments. The comments are all wrong and you can't see the real code through the comments. How do you disable comments for a while? Make them white!

1. Select *Options/Hypertext options*.
2. Select *Comment*.
3. In the color palette, click the white color box, then press *Save*.

All comments are suddenly invisible! Is the code any better now? To bring the comments back, do this:

4. Select *Options/Hypertext options*.
5. Select *Comment*.
6. Click *Default*, then *Save*.

10. Classify windows and dialogs

What exactly are the dialogs, toolboxes and other windows in your program? Where are the forms with menus and which forms have a Default or a Cancel button? The *Form report* tells you that. Note that this report works with classic VB only. There is no support for VB.NET or VBA.

In the *Report* menu, select *Form report*. Here is a sample report and how you read it:

| Form | Cbox | Min | Max | Border | Menu | Btn | MDI |
|----------------|------|-----|-----|---------|------|-----|-------|
| frmAboutBox | Y | - | - | FixDial | - | DC | - |
| frmAddField | Y | - | - | FixDial | - | DC | - |
| frmAddIndex | Y | - | - | FixDial | - | DC | - |
| frmAttachments | Y | Y | Y | Sizable | - | -C | Child |
| frmCopyStruct | Y | - | - | FixDial | - | DC | - |
| frmDatabase | Y | Y | Y | Sizable | - | -- | Child |

| | |
|----------|---|
| Cbox | This column tells you whether the form has a ControlBox. If it has one, it should also have a custom icon. You might want to take a look at the icon in VB and verify it looks good. |
| Min, Max | These columns tell you if there is a Minimize and a Maximize button on the form. |
| Border | This is the BorderStyle property. |
| Menu | This column displays Y if the form has a menu bar. It shows H if there are only hidden menus, which are probably used for popup menus. This way you can find the main form(s) in your application and also those dialogs that might define popup menus. |
| Btn | This column displays D for Default button and C for Cancel button. Each dialog should preferably have both. Other forms should probably have neither. |
| MDI | This column shows MDI Parent and Child forms. |

11. Detect side effects of out parameters

When a procedure changes the value of a ByRef parameter, the changes are propagated back to callers. This can cause nasty side effects when a variable changes unintentionally. A ByRef parameter whose value changes is called an *out* parameter. In most cases, the function return value is a more appropriate way to return a value to callers.

Project Analyzer can list the out parameters for you to review:

1. Select *Report/Module members...*
2. Select *Procedures with out parameters.*
3. In the *Parameter* style combo box, select *Detailed with param types* or *Detailed with param names.*
4. Press *OK.*

12. Tweak Project Analyzer

Sometimes Project Analyzer is not giving you optimal results because of some suboptimal coding you need to use. As an example, you need to use late binding, but Project Analyzer would work better with early binding. You wish you could tell Project Analyzer more about your code, but you can't simply go on and change the way your code works in VB.

Here's the solution. Project Analyzer internally defines a special compiler constant `#Const PROJECT_ANALYZER = True`. You can use conditional compilation to define something just for analysis purposes. Here's an example:

```
#If PROJECT_ANALYZER Then
    Dim x As MyForm ' Early binding, easy to analyze
#Else
    Dim x As Object ' Late binding, hard to analyze
#End If
```

Conclusion

You can use Project Analyzer in many ways not apparent at first sight. It pays off to examine the [help](#) to find hidden features and new ideas. Examine the menus, right-click every list control for context menus, press the toolbar buttons. If you find a new way to use Project Analyzer for a practical task, feel free to tell Aivosto about it. It could well be worth documenting and sharing with other users.