VBPro - DoDi's VB 3.0 Project Manager

The project manager **VBPro** assists you in the creation and maintenance of applications written with VB 3.0, simplifies the usage of general code modules in multiple projects, and protects programs against Discompilers.

Imagine, you write a large application and want to reuse existing modules. For debugging purposes you need breakpoints, dumps and messages, that shall not occur in the final program. For public relations, you need a somewhat restricted or modified demo version of your application. All registered users must receive updates, and the programs shall be protected against Discompilers. After the great success of your programs, you want to create nationalised versions in other languages.

Exactly for these requirements **VBPro** was developed, and more features may be added on demand.

Usage of VBPro

With VB, you create a new project, adding existing modules, or use an already existing project. Then you pass the **make file** (*.mak) to **VBPro**, that creates a **project file** (***.pro**) from it.

All modules should be saved **as text**, to allow pre-processing of the sources.

The modules that you want to use in multiple programs may need modifications and extensions, but they still must be working with all your applications. To prevent all modules from inadvertent changes, the project manager creates a **new directory** for every version of a project, copies the sources into it and creates a make file with the new location of all modules.

With Project|Start|Original from the menu you run the interpreter with the base version of your program. There you mark all parts of the sources that will be different for distinct versions of your program. How you do this is described later (**filtering**).

Normally you'll need at least one final version of your program without debug code. Therefore you declare a global variable (e.g. Debugging, Test...), that is checked in the program with **If** to execute the debug code only if the variable is set to True. You also outdent all these conditional statements (**If / Else / End If**) to the first column, so the pre-processor built into the project manager can detect and execute these statements.

Now you test your program in the original version with the interpreter, setting the conditional variables as required for the different versions of the program. In case of trouble, you simply activate the debug code by setting the appropriate variable, and bring the program to the point where an error occurred. You may also test different algorithms by setting the appropriate variables as required.

After this you create a specific version of your program with the Project|Create|Conditional option of the project manager, and start it with either the MsgBox coming up after the files are copied, or with the Project|Start|Copy option. If that version needs further modifications, you overwrite the original modules with the modified versions, or use **VBDiff** to detect all changes and edit them in the original files.

The project manager must know the settings of all conditions for the different versions of your program. The values cannot be taken from the sources, because there stands at most one fixed value, inappropriate to create different versions. Therefore the project manager asks you with every **unknown** conditional variable, which state should be assumed for it, and enters that value in the project file.

If you want to create **other versions** of the program, you select an existing version in the project list, enter a new version name, and with the New button you create a copy of it with the new name. In the new version, you modify the conditions as desired and create that version as shown above. You can use the same name and directory for the different versions, because at any time all versions can be derived from the original files.

In the interpreter, you test all versions and compile the programs. After changes, you quickly rebuild

all executables with the option Project|Create|EXE, and test the compiled programs with Project|Start|Program. To retain each executable, you should compile them into another directory, because the project directory is always erased before copying the sources into it.

When you ship the new versions to all registered users, you're assisted by the **user database** built into the program manager, and the specific **version information** you may compile into every copy of the executables. You reach both functions with Options|Version, whereupon a window with the version information will pop up. For the first time, you may want to use an existing version resource, that you load with Resource|Load, modify it as appropriate and save it with Resource|Save under the name marked in the project manager's main window.

In the User menu you enter new users with Register, and with Select you select a user, who's name is copied into the version resource. Then you create with User|Diskette a disk with the program, that contains the version information and some modifications that will cause trouble to any Discompiler, and all the files you added in the project description. The professional version will also handle large files that don't fit on a diskette.

Details

While the preceding section described the development process, here you'll find the detailed features of the project manager **VBPro**.

The Menu

Menu Project
Here you find all functions related to a project.

Project|New...
... creates a new project file, based on an existing make file (*.MAK).

Project|Open...
... opens an existing project (*.PRO).

Project|Save
... saves the current project under the path and name given in the project field. You may edit this field to save the project file in a different place.

Project|Translation
If this option is checked, all text strings in forms are translated while the sources are copied, based on a translation table.

Menu Project|Create
With this menu you create different versions of the program you selected in the project list.

Project|Create|Copy
... copies all files to the directory given, without modification. This goes very fast, you may use this option to create a reference project that allows the detection of all changes you may implement while testing the program.

Project|Create|Conditional
... copies the sources, excluding all parts of the code with False conditions.

Project|Create|Debug
... copies the sources, converting the code with False conditions into remarks. This allows you to search in the interpreter for subroutines and other parts, inadvertently marked for exclusion in this version of the program, and re-enable code, type declarations and so on. For running and compiling, there is no difference from the version created with the option Project|Create|Conditional.

Project|Create|EXE
... creates the executable file. The interpreter is called, compiles the program and terminates. After

this, you can start the program and create diskettes.

Due to an **error in VB 3.0**, the path and file name of the executable (in the make file) is ignored with this option! Check if the program is created in the desired location, or better use Project|Start|Copy and compile the program manually.

Project|Create|Disk
... copies all files given in the version description to drive A:, but excluding all source modules. The version **Resource** (if given) is compiled into the EXE file.
While copying the EXE file, some **unneeded informations** (e.g. names of controls, Option|Remove Names) are **removed**, and other modifications are made to **protect** it against Discompilers.

Only give away programs protected with this option!

Protection is **not** implemented in the **Demo** version of **VBPro**!
In the **Pro** version the files are copied to another subdirectory, supporting projects that cover more than one diskette.

Menu Project|Start

Project|Start|Copy
... runs the copied version in the interpreter.

Project|Start|Original
... runs the original version in the interpreter. Use this option to make changes required in all versions of the program, to rearrange Type declarations, global variables and subroutines into different modules, including changes in modules common to multiple projects.

Project|Start|Program
... runs the compiled executable.

Other Options in the Project Menu

Project|Info...
... shows the version of the project manager, useful for error reports and suggestions to the author.

Project|Exit
... exits the project manager. If changes in the project are not yet saved, a reminder pops up and you can choose to save or abandon the changes, or to resume project manager.

Menu Options

With this menu you set options for the project manager, or add files to the project.

Options|Data...
... shows a list of the original directory, from which you can add files needed while the program is running.
In the mask field you enter the search pattern for the files (*.DAT, *.TXT...). Files are added to the project list with a double click into the file list, with the button All all files shown are added.
The files will be copied later with any Copy option to the working directory or to the diskette.

Options|Extra...
... does the same, but the files will be copied only to the diskette. This option is useful for documentation and other files you want to ship with your programs, but which are not required for running the program for testing and debugging.

Options|Version
... shows the version resource window for modifications and maintenance of the user database.

Options|Shift Popup
... determines whether the popup menu in the project grid comes with or without pressing an

additional key (Shft, Ctrl). In the other case, you can adjust the height and width of the cells in the grid.

Menu Protect

This menu allows the protection of any compiled VB3 program against decompilation.

Protect|Remove Names
... determines whether the names of controls are removed while copying the executable file. The documentation states that these names are not available at runtime, and until now no problems were reported with this option. Therefore it is normally enabled, but as a precaution I made it selectable. Errors were reported in protected programs using SBC.VBX, this must be fixed in a future release.

Protect|*.EXE...
... pops up the **Protection** window, where you enter with the buttons the compiled Program, optionally a VersionInformation resource, and then the Destination path and filename for the copy of the program. While copying the EXE file, some **unneeded informations** (e.g. names of controls, Option|Remove Names) are **removed**, and other modifications are made to **protect** it against Discompilers. The version resource (if given) is compiled into the EXE file.

Protection is **not** implemented in the **Demo** version of **VBPro**!

Fields in the Main Window of the Project Manager

On the top you find the **project field** with the path and name of the current project file. This field is used whenever the project is saved. You can edit this field before saving to create a new project file, but never change the extension (PRO).

Below this comes the **version field** with the buttons New, Modify and Delete.
The **version field** shows the name of the version selected from the project list. You may enter a new name and then Modify the version name or create a New version.

The New button creates a copy of the selected version with a new name.
The Modify button changes the name of the selected version.
The Delete button deletes the selected version from the project.

The next line contains a **type list** from which you can select the type of an entry. The name of the currently selected entry in the project list is displayed in the edit field beneath.

The **type list** contains the names of the fixed entries (mostly located in the MAK file), and the types applicable to files and conditions.

File types are:

type	**the file is**
SourceFile	entered in the make file and filtered during copying
VBX/DLL	entered in the make file, but not copied
DataFile	copied, but not entered in the make file
Extra	copied only to the diskette, for special VBXs, DLLs, documentation...
Never	excluded from that version of your program

Condition types are:

type	**with this condition, the code**
Always	and the conditional statements are copied without modification
Never	and the conditional statements are excluded from the copy
True	is put into the copy
False	is excluded from the copy

The buttons below this have the following functions:

The New button adds a line to the project list, with the given text and the type selected from the type

list. Already existing files or conditions are only updated.

The Modify button updates the entry selected in the project list with the given text and type. Pressing Enter in the text field has the same effect.

The Delete button deletes the selected row from the project list. Two cases may exist:

1. The row contains fixed information, or the project list contains more than one version. Then the entries in the selected row are initialised to the appropriate default. The row can be completely removed only with the popup menu.

2. There exists only one versions, and a file or condition is selected. Then the row is completely removed from the project list.

The bottom of the window is always completely filled with the **project list**, depending on the size of the window. This list is implemented with a **Grid** control, refer to the VB manual for details about it.

The project list starts with predefined entries, related to the make file of the different versions. These lines are followed with the files belonging to the project, up to the conditions at the end of the list. Every version of the project is described in a distinct row of the project list, with the files and conditions appropriate for that version.

With a **click** on any field in the project list, a **popup menu** appears where you can comfortably set the conditions for the selected entry, edit a file or delete the whole line. Unfortunately, there is a conflict between the popup menu and the adjustment of the cells' dimensions. To adjust the cells, press a control key (Ctrl, Shft) to prevent the popup menu from coming up. You may reverse this procedure with the Option|Shift Popup.

Conditions (Filtering)

While copying source modules (type **SourceFile**), the project manager interprets all lines without leading spaces, that start with

If, ElseIf, Else, End If

In these lines, the pre-processor evaluates the conditions and accordingly copies or suppresses (filters) the corresponding range of statements.

Code after **Then** in an **If** or **ElseIf** statement is ignored, only **block-ifs** with a corresponding **EndIf** are allowed.

Conditions in **If** and **ElseIf** can only contain the name of a variable or constant, expressions are **not** evaluated by the pre-processor!

The syntax is fully compatible with the interpreter, but there may be cases where ranges should be filtered, that cannot be enclosed with If...End If (in the declarations section, Select...). Then you may hide the conditions from the interpreter in a remark (with an apostrophe **'**). After that character no spaces are allowed, else the line is treated as a comment. You may also omit **Then** from these lines.

The names of the conditions may be constants or variables in the source code, they also may be completely undefined there, but then the project can be run in the interpreter only after the project manager has removed these conditions. The pre-processor looks up the state of the conditions only in the project list, assignments in the source code are ignored.

Whenever the pre-processor encounters an undefined condition name, a MsgBox is shown where you can enter the appropriate value. With the Cancel button, processing of the file is stopped, so that you can find where the statement occurs in the source file (the name of the current file is displayed in the capture of the project manager window).

Possible values for conditions are **True** and **False**, or **Always** and **Never**.

With **True** and **False**, filtering occurs as in the interpreter, only the lines with the condition and the code in the False part of the conditional statement are removed from the copy.

With **Always**, the code is copied without modifications, so that it will be executed by the interpreter. Then you may set the variable manually or by a menu option while running the program, in another version of the program you may declare a fixed value for the same condition. Accordingly, with **Never** the code is completely removed from the copy.

Always and **Never** can be used only in the first **If** of a conditional statement, the following parts (ElseIf, Else...) are not interpreted and only copied (with Always) or removed (with Never). However, other (nested) **If**s in the copied range are filtered by the pre-processor as usual.

Errors may be reported while a file is filtered, when there is no **If** for an **Else** or **End If**, or when at the end of a module one or more **End If** are missing. Then you may cancel the action and examine the source file, that will be copied up to the line with the error. If you don't use hidden conditions (**'If**), such errors are also reported in the interpreter when you try to run the program.

In the final version of your program, there should be no **Stop** and **Debug** statements. Such lines are shown in a MsgBox and you can choose to copy or omit the whole line, or to cancel the action (as described above).

Examples for Filtering

The differences between the evaluation of conditions in the interpreter and filtering with the pre-processor are faster execution of the program, because the evaluation occurs only once in the pre-processor, but more interesting is the effect, that variables and functions used only in the removed statements can be omitted from the program! You can move all these variables and subroutines to distinct modules, that must be included only in those versions of your program, where they are really used. You may also implement different algorithms, and select later which one shall be used. In the original source, all variations are retained and may be used in other versions or projects.

In the following examples, the lines handled by the pre-processor are written **bold**.

Example:

```
If Debugging Then
  testsub a,b,c ' this function must not occur in the final release
  Debug.Print "something"
  Stop ' Stop should not occur in the final release!
ElseIf German Then
  MsgBox "irgendein Fehler'
Else
  MsgBox "any error"
End If
```

Hint:
Programs can be translated to different languages with string constants (Const) instead of constant strings ("..."). For every language, you create a separate module with the strings translated to the respective language as Global Constants. In the **BaseFile** option you enter the module appropriate to the desired language. More support for the nationalisation may be added (see Project|Translation).

More effects with filtering are possible with hidden conditions, written like usual but hidden from the interpreter in remarks. Hereby you can exclude parts from declarations or Select statements, where the interpreter does not accept conditional statements.

Example:

```
'If Debugging Then
Type debugrec
 f(100) As Integer
'If FixStrings ' Then can be omitted
```

```
  s As String*256
'elseif NoStrings
  ' nothing is defined
'else
  s As String ' bad example, in the original source the interpreter sees a "duplicate variable"
'end if ' Strings
End Type
Dim d(1000) As debugrec
'end if ' Debugging - such a comment improves readability of your sources
.
.
.
   Select Case x
   Case 0: ...
   Case 1: ...
'If Debugging
   Case 2: Stop ' should never occur
   Case 3: Stop
'End if ' Debugging
   End Select
```

You can also filter entire subroutines, but the behaviour of the interpreter discourages that.

Example:

```
'if debugging
Sub abc
...
End Sub

'end if
Sub xyz
...
```

In this example, the interpreter keeps the If line with Sub abc, but the End If line with Sub xyz. Whenever one of these routines move in the source file, the result is unpredictable, but always undesired and therefore wrong. You can safely add such comments only with an editor in the source files, but every change of that module in the interpreter is hazard. You better move all functions, that are needed only in some versions of your program, into separate modules. In the program manager, you can exclude these modules by retyping them from **SourceFile** to **Never** as appropriate for the different versions.

Version Information

You can add a **version resource** to the final versions of your programs, that the file manager of Windows for Workgroups can show, and that can also be used by the VER.DLL during installation. In the project list you can enter any resource file as Resource, but the file should contain at least a version information resource. To create, examine or edit the version resource, choose Option|Version from the menu, or Edit from the popup menu. Once the Version Information window is open, you can load another resource there.

The meaning of the fields in the version resource is described in the WIN31API.HLP help file. With the option User|Selection, the **SpecialBuild** field is filled with the name of a user from the user database of the project manager. This should eventually be the **PrivateBuild** field, I choose the field based on the display of the file manager in the German version. With the project manager, it will be no problem for me to change this in the English version or all versions together ;-)

You can add the version information in **two different languages**. First you must enable that feature by checking Resource|2 Languages, then you can switch between both versions with the radio buttons beneath the language options. The implemented switch determines which language is really

implemented in the program. I never found a single program really usable in different languages, so I think the VER.DLL should be able to detect, which language is really supported. The modify button is **not** needed with a **single** language resource, but before switching between two languages with the radio buttons, you **must** save changes with the modify button!

With User|Registration you reach the **user database** with all your products and all users. If no USER.REG file is found in the project directory, you can select an existing database elsewhere or create a new database. Then you can enter all your programs into the database, then add the users you should send updates, and which programs they shall receive. More features may come with the professional version of the project manager, like creating disk labels and address stickers for shipping.

Like with Project|Create|Disk in the main window, you can create series of diskettes from the version information option User|Diskette. The checksum of the version resource is included in the program, so manipulations on a program can be detected and illegal copies traced back to the user who made the copies. Also, a program copied with this feature is protected against Discompilers like **VBDis**.

With User|Harddisk, available in the **Pro** version, the files are copied to another subdirectory, supporting projects that cover more than one diskette.

Hints:
For multiple programs, you should prepare multiple resources with your copyright, your products, programs and the multiple versions of your programs. Then with Resource|Load... you can load the appropriate resource as base for a new product, program or version. If no version resource exists, a default resource is created. Then you fill in the required fields, using the tab key to cycle through the most important entries. If you save the resource, either with Resource|Save or when closing the version resource window, the resource is written to the file currently selected in the main window of the project manager; to write the resource to a different file, select or edit the name there. If your product consists of more than one program, you should use a unique working directory name in each project file, then the programs are copied all to the same subdirectory on your diskettes.

Nationalisation of a Project

VBPro supports the translation of programs into different natural languages.

For strings in the body of the modules, you may use string constants and put the corresponding strings into different **BaseFile** modules, and use translated version **Resource**s for each version of your program. To translate the strings contained in forms, check Project|Translation before creating a copy of your project. **VBPro** currently supports only one additional language, this will be changed in a future release.

If the Translation option is checked, the **translation** window pops up while copying the source files. On the first call, the translation table for the project is created and displayed in a grid control. There you enter all the strings that shall be translated. Strings that need no translation may be left empty. The focus is normally kept in the edit field above the table and some keys have a special meaning if pressed in this field:

Enter	saves the string and updates the table
Cursor up	selects the previous entry in the table
Cursor down	selects the next entry in the table

You can also select an entry by clicking into the table.

When you close the translation window, the table is saved to a file with the extension **trx**, from where it is retrieved whenever needed. **The table is not saved in the demo version!**