DoDi's Visual Basic Tools

(State October 1995)

These tools shall help develop programs with Microsoft's Visual Basic 3.0 (VB). They group around a project management system, with extensions for the optimisation of programs.

I missed the following functions in VB and started to write my own tools for the following tasks:

- Viewer for modules saved in binary format
- Crossreference of data types, variables and subroutines
- Conditional compilation of the sources
- Version information in the executables
- Nationalisation of applications
- Comparison of source modules
- Management of all files needed to develop and execute a program
- Optimisation of programs
- Protection against Discompilers
- Protection against setup programs

- Resource editor for executables
- Protected source modules
- Conversion of applications to new VB versions
- Translation from VB into other languages (C++) to simplify the development of DLLs

Now the following VB-Tools exist:

VBDis3	Discompiler for VB 2.0/3.0
VBMDis3	Optimisation tool for VB 2.0/3.0
VBCtrl2	Custom control analyser (for the Discompiler)
VBDiff	Comparison tool for VB modules
VBMap	Project browser, viewer for binary modules
VBPro2	Project manager and pre-processor
VBXlock	Protection for system configuration and VBX files against setup programs
SetDown	Application installer, bypassing the setup programs

The following files contain detailed descriptions:

File	**Programs**
STATE.WRI	last minute changes
VBTOOL_E.WRI	overview (this file)
REGISTER.WRI	license information and registration form
VBPRO_E.WRI	**VBPro**
VBMAP_E.WRI	**VBDiff, VBMap**
VBDIS_E.WRI	**VBDis**
VBMDIS_E.WRI	**VBMDis, VBCtrl**

All programs were written by **Do**ktor H.-P. **Di**ettrich (**DoDi**). The restricted **demo versions** may and shall be copied and distributed. Registered users receive the unrestricted versions for their own use, refer to REGISTER.WRI for details.

The **names** of the files may differ from the documentation. Some programs have a **digit** appended to their names to denote new and incompatible versions, **English** versions have an appended **'e'**.

Some of the shareware programs are **Lite** versions, with less features than the according professional versions. The **professional** versions are **not** distributed as shareware and are not yet available.

Due to the demand for the professional versions, a **preliminary upgrade** is available until the professional edition becomes available. This upgrade consists only of the enhanced programs, no other features are available (online help, manuals...).

Installation

The programs need no special installation, no modifications are required in your INI files. Simply copy all the files into your VB directory or a new directory (recommended for the Discompiler with the many data files). If you got archives, unpack these into the desired directory.

Now to the description of the already implemented features:

Conditional Compilation

With VB it's hard to develop modules for use in different applications. Changes made for one application may cause malfunction in other applications. We need some mechanism to create different versions from the same sources, that can be accordingly selected for every application. This concept includes different versions of the same application (restricted/unrestricted...), and nationalised versions.

State:
The project manager VBPro contains a pre-processor, that recognises simple conditional expressions (**If/Else/ElseIf/EndIf**) and omits the False parts of the blocks. The conditions can also be evaluated at runtime, so it is possible to test all branches and to switch debug code on and off. The pre-processor can be instructed to copy either the selected parts only or the whole conditional blocks to the new source files. Nested conditions are implemented, too. Undefined conditions are trapped and the user can enter the appropriate value in a dialogue box.

Version Comparison

It may be difficult to detect all changes made to VB source modules, because the interpreter may rearrange subroutines in the source files, the case of names may change and so forth. This behaviour must be considered by an appropriate file comparison tool.

State:
VBDiff compares text files and shows differences in two windows. The search must be restarted with two identical lines marked by the user. For rearranged subroutines, a double click on a Sub or Function definition searches for the same routine in the other file and resumes the comparison. Parts of the sources shown can be copied to the clipboard.

Protection against Discompilers

The analysis of the EXE files created with VB 3.0 has revealed a lot of informations from the source files, that are intentionally put into the executables. These informations don't affect program execution, but they make it very easy to reconstruct the sources with many details, like the exact names of every control and the indentation of all lines. If it is so easy to look into any program, there may exist more than one Discompiler, so some protection is needed against such tools.

State:
Provisions are implemented in the **project manager**, to remove the names of all controls and do some other changes while copying the executables to a diskette.

Crossreference

With big projects, the declarations of variables and data types may be spread over many modules. VB allows to jump directly to the definition of a subroutine, but other informations can be found only with a full text search. VB also hides the path of the modules of a project, and sometimes modified modules are placed in different directories without notification.

State:
VBMap analyses the modules of a project and displays a list of all subroutines, variables, constants and user defined Types. From that list you can directly go to the source text where such a symbol is defined.

Viewing Modules

In the development environment, access to modules from other projects may be impossible, if these are saved in **binary** format. It should be possible to view any module at any time, and to copy parts of it to other modules.

State:
VBMap shows modules just like the interpreter does, even if they are saved in **binary** format. Selected lines can be copied to the clipboard.

Project Management

The project manager **VBPro** allows to create different applications based on shared source modules. Even from one make file, different versions of an application may be created.

These steps may be necessary to create an application:
1. Create a project with the interpreter
2. Test the new application
3. Add data files required to run the program
4. Create customised versions, using conditions and shared source modules
5. Test the customised versions
6. Add required changes to the shared modules
7. Remove debug code
8. Compile the program
9. Create nationalised versions
10. Create diskettes with the protected program

Here is what you do in these steps:

1. Create a project with the interpreter
The program is developed as usual with the interpreter. The shared modules can be configured with global conditions (variables or constants).

2. Test the new application
Test the program with the interpreter, setting the conditions as appropriate to cover all versions of the program. Debug code may be added freely, but it should be flagged for omission from the final versions. Use separate forms and modules for functions that are used only while debugging the application.

3. Add data files required to run the program
Pass the make file of the application to the project manager to create a project description. Add data files and other documents to the description. Define the appropriate conditions for the pre-processor.

4. Create customised versions, using conditions and shared source modules
The project manager copies all required files into a new directory, removing parts of the source code as defined by the condition settings.

5. Test the customised versions
With parts of the sources conditionally omitted from the sources, the interpreter may encounter undefined variables and subroutines. Rearrange the missing parts in the original sources, until needed and unneeded parts are clearly separated.
Minor changes can be done on the fly. The pre-processor can be instructed to copy all excluded lines from the sources into comments, so you can find all missing declarations, and uncomment them.

6. Add required changes to the shared modules
If in the previous step changes were made in the copied sources, you must introduce these changes to the original files. Best you rename the directory with the changed files, and create the same project again as reference. Then start **VBDiff** to compare both versions, and copy the changed parts to the original sources (using the interpreter or any editor).

7. Remove debug code
In the preceding steps you may have used debug code to test the program, including forms to view dumps or other stuff only needed while debugging. Now you create one ore more final versions of your project. In the project manager, copy the debug version, change the conditions to exclude the debug code, and mark all modules that shall be excluded from the final programs.

8. Compile the program
With the project manager, create the final versions and compile them with VB. If errors occur due to the omitted debug code, correct this as shown in step 5.

9. Create nationalised versions

Translating a program to different languages requires some changes to the sources. Replace all text strings with string constants, and collect them in a single module as global constants. Translate all strings in this module and save it under a different name, which you enter as **BaseFile** in the project description.

The strings in caption and text properties of the forms and controls can be replaced with the Translation option of the project manager. All these strings are displayed in a grid control, where you can enter the translated strings in a separate column.

10. Create diskettes with the protected program

Now the project manager can copy the program and the additional files (documentation, DLLs...) to diskettes. While copying the executable, changes are applied to make it smaller and better **protected against illegal copies and Discompilers**. You may also add a **version information** resource with the name of the user, which you retrieve from the integrated **user database** of the project manager.

Optimising your Programs

An optimising tool shall detect unused variables and subroutines, and make suggestions to improve the performance of the program. Unneeded information must be removed from the executable, making it smaller, faster and better protected against Discompilers.

State:
VBMDis helps to optimise your source code. The tokens of the source modules and the executable are displayed, so you can find unwanted type conversions, and unused variables or subroutines. You can apply the appropriate changes to your source files.
VBPro removes unneeded names from the executable, making them significantly smaller. Together with other modifications, the program will better withstand attacks from a Discompiler.

Optimising with Native Code

VB programs can run considerably faster, if time consuming calculations are done in native code in a DLL. Required is assistance for writing the code for a DLL, or direct inline compilation of parts of a program. Even VB 4.0 has no such features.

State:
It is possible to display VB modules in C++, but this is not yet implemented. Similar programs are announced by several suppliers, so my priority for that feature is low.

However, newest investigations revealed a chance to recompile parts of the executable, it should be possible to include the native code directly into the program. It might be possible to optimise any existing application this way!

Protection against Setup Programs

When I had to re-install VB for the third time, because a setup program had overwritten VBX and DLL files with old versions, and modified CONFIG.SYS, AUTOEXEC.BAT and INI files that I couldn't even boot my system, I sat down and wrote some tools to outwit such viral programs.

State:
VBXlock absolutely protects the standard VBXs and DLLs of VB 3.0, and prevents changes to other system files.
SetDown is a dumb installation program, that expands all the files shipped with a VB application to a single directory. After this, most applications can be run and easily removed by deleting that directory. If the application won't run without an installation, I use **VBDis** to create the sources from the SETUP1.EXE and run it in the interpreter. There I can skip over all unwanted steps, an in-depth analysis of the program is not required. For older versions of VB, the setup programs are shipped as source files, so **VBDis** is not needed.

As long as Visual Basic has similar deficiencies in its setup kit, I'll publish Discompilers for all new VB versions, too, that at least can crack the setup programs.

The following features are not yet implemented. There was no demand for them till now, therefore the development is delayed until VB 4.0 is available and analysed, then the priorities of these projects are rearranged.

Resource Editor

Many programs are not designed to run with monochrome (LCD) displays or a different graphics resolution. It should be possible to extract the forms form the executables, edit them in the interpreter and compile them back into the executable.

State:
VBDis creates all forms from an EXE file. The opposite direction is possible, too, but with the sources the program can be rebuilt as well.

Converting Old VB Applications

Executables from earlier VB versions (using VBRUN100/200) should be converted to VB 3.0, to overcome errors in the old libraries.

State:
The file formats of the executables are partially known. VBDis3 also handles VB 2.0 executables, so these can be recompiled with VB 3.0. For VB 1.0, there seems to be no great interest in such a feature. A conversion from VB 2.0 to 3.0 seems to be possible on the EXE level, too.