VBMDis - *der* Make-Discompiler

VBMDis ist der direkte Nachfolger von **VBDis**, dessen Entwicklung nunmehr (hoffentlich) abgeschlossen ist. **VBMDis** dient vorrangig der Optimierung von Programmen und erlaubt zunächst den **Vergleich** der Quelltexte mit dem ausführbaren Programm, so daß Sie selbst anhand der **angezeigten Tokens** und Beschreibungen der Variablen feststellen und ausprobieren können, mit welchen Änderungen Ihre Programme kleiner und schneller werden.

Die Installation von **VBMDis** ist wie bisher ganz einfach, dazu müssen Sie nur das ganze Verzeichnis **VBDIS** auf Ihre Festplatte kopieren. Sie werden nicht von irgendwelchen Setup-Programmen gequält und brauchen keine Angst um Ihre INI-Dateien, VBXe und DLLs zu haben, dort werden keinerlei Änderungen vorgenommen! Die Deinstallation ist genauso einfach, dazu löschen Sie das Verzeichnis wieder, und der Discompiler ist restlos von Ihrem Rechner verschwunden.

Erzeugen Sie das ausführbare Programm und geben Sie **VBMDis** den Makefile an. Dann startet der Discompiler (wie in VBDis), danach erscheinen zwei Fenster mit den Modulen des Interpreters und des ausführbaren Programms. Die Fenster sind jeweils geteilt in die Auswahl des Moduls und des Unterprogramms sowie die Anzeige des Textes und der Tokens für die aktuelle Zeile. Die Größe der beiden letzten Bereiche können Sie ähnlich wie im Interpreter einstellen, der Mauszeiger ändert über den hierfür vorgesehenen Bereichen seine Form entsprechend ab.

Für die Anzeige der Tokens müssen die Quellen unbedingt binär gespeichert werden.

Wenn Sie Ihre Programme normalerweise als Text gespeichert haben (kompatibel mit dem Projekt-Manager), erzeugen Sie am besten eine Kopie des Projekts und speichern alle Module binär ab. **VBPro** unterstützt dies mit der Option Projekt|Erzeugen|Kopie, nur die Speicherung als Text müssen Sie selbst durchführen.

Die Fenster von VBMDis

Das Hauptfenster ist ein MDI-Fenster, in dem nach dem Öffnen eines Projekts zwei Fenster für die Module des Interpreters und des compilierten Programms erscheinen. Für die Analyse des Programms erscheint das von **VBDis** her bekannte Fenster. Zusätzlich können Fenster mit den globalen Variablen und den Modul-Variablen geöffnet werden.

Mit der Option Fenster|Untereinander können Sie wählen, ob die Make- und Exe-Fenster im MDI-Fenster nebeneinander oder untereinander angeordnet werden sollen. Lange Zeilen lassen sich besser lesen, wenn die Fenster untereinander (und damit breiter) angeordnet sind, während der Vergleich der Texte und Tokens einfacher ist, wenn die Fenster nebeneinander angeordnet werden.

Die Auswahl der Module und Unterprogramme erfolgt in beiden Fenstern getrennt, im Exe-Fenster wird zusätzlich ein Modul mit allen globalen Deklarationen angezeigt. Zu der im Quelltext markierte Zeile werden im darüberliegenden Teil des jeweiligen Fensters die zugehörigen Tokens angezeigt, mit Namen und den dazugehörenden Argumenten (hexadezimal). Zu den Deklarations-Teilen gibt es keine Exe-Tokens, auch nicht zu den Deklarationen der Unterprogramme und der lokalen Variablen. Dafür gibt es im ausführbaren Programm einige Tokens, die vom Compiler eingefügt werden und in den Quelltexten nicht auftauchen; diese Tokens sind meistens am Namen **exe** zu erkennen, zusätzlich können Typ-Konvertierung (**C<typ>**) hinzugefügt werden.

Zusätzlich wird im Exe-Fenster dargestellt, welche Argumente im **Stack** liegen. Als Typzeichen werden neben den Kennzeichen von VB (**%&!#@$**) noch **A** (Arrays), **O** (Objekte), **T** (Type) und **v** (Variant) verwendet. Pointer sind an einem Control-Zeichen (**^**) vor dem Typ des Arguments erkennbar, wobei Felder, Types und Objekte immer als Pointer abgelegt werden. Strings werden

ebenfalls immer als Pointer abgelegt, ByVal erzeugt hier einen Pointer auf eine Kopie des Strings. Deshalb kann der Discompiler nicht immer unterscheiden, ob Strings ByVal oder ByRef übergeben werden, dies ist nur am Aufruf des Unterprogramms erkennbar.

Die Argumente bzw. Operationen des Tokens werden in der nächsten Spalte nochmals aufgelistet, wobei eine Funktions-ähnliche Notation verwendet wird. Der Datentyp, der vom jeweiligen Token im Stack abgelegt wird, wird dabei ebenfalls als Typzeichen dargestellt, davor kann noch ein Hinweis auf die verwendete Variable stehen. Unterprogramme und Variablen werden mit Scope und Offset dargestellt, mit den Zeichen g, m, p und l für **g**lobale und **m**odulspezifische Unterprogramme bzw. Variablen sowie **P**arameter und **l**okale Variablen. Bei Parametern kann auch die Kombination pv für By**V**al-**P**arameter auftreten. Danach kommen in Klammern die Argumente, welche bei der Ausführung des Tokens vom Stack genommen werden.

Die Darstellung des Stacks erlaubt die Erkennung überflüssiger Typumwandlungen, hierzu gleich ein Beispiel mit **Mid** und **Mid$**:
Die Zuweisung

```
s$ = Mid$(x$, y%)
```

erzeugt direkt einen String, während

```
s$ = Mid(x$, y%)
```

zunächst auch einen String erzeugt, weil im Laufzeitsystem nur **Mid$** implementiert ist; dieser String wird dann nach Variant konvertiert (Notation **v($)**), was dem Ergebnis von Mid() entspricht. Vor der Zuweisung wird dieser Variant wieder in einen String umgewandelt (Notation **$(v)**).

Bei Konstanten und Single-Variablen können ebenfalls überflüssige Konvertierungen auftreten, die das Programm unnötig verlangsamen. Die jeweils beste Variante können Sie am einfachsten ermitteln, indem Sie diese in aufeinanderfolgende Zeilen einbauen und die erzeugten Tokens vergleichen. Der Typ Single ist eigentlich nur in Feldern sinnvoll, wo Speicherplatz gespart werden kann, bei den meisten Zugriffen erfolgt dann eine Konvertierung in Double (Notation **#(!)**), die das Programm sicher langsamer macht als der direkte Zugriff auf eine Double-Variable.

Buttons in den Code-Fenstern
Da die Reihenfolge der Module im Makefile und im compilierten Programm unterschiedlich sein kann, müssen Sie die Zuordnung der Module und ggf. auch der Unterprogramme selbst vornehmen. Hierzu dienen die beiden Buttons **zuweisen** im Make-Fenster, der obere kopiert den Namen des Moduls und aller Unterprogramme in das ausgewählte Modul im Exe-Fenster, der untere Button kopiert den Namen der ausgewählten Funktion, sofern dies notwendig sein sollte.

Die übrigen Buttons im Make-Fenster sind eigentlich Debug-Funktionen für den Discompiler, ich habe sie vorläufig noch dringelassen, damit Probleme besser zu diagnostizieren sind. Wenn sie diese Funktionen ausprobieren wollen, aktivieren Sie zunächst Modus|Vergleich im Menü. Daraufhin werden alle Namen durch **<var>** ersetzt, und im Make-Fenster werden alle Zeilen unterdrückt, die im ausführbaren Programm nicht erscheinen. Dann können mit den Buttons wahlweise alle Unterprogramme, das aktuelle Unterprogramm oder das nächste Unterprogramm verglichen werden.

Im Exe-Fenster können sie mit dem Button **umbenennen** den Namen eines Moduls vorgeben, den Sie in das Textfeld der ComboBox für die Module eingetragen haben. Der Button **Typen?** aktiviert ebenfalls eine Debug-Funktion, die zur Diagnose einiger Probleme bei der Feststellung der Variablen-Typen verwendet wird. Interessanter ist der Button Variablen, der ein Fenster mit den Variablen des aktuellen Moduls öffnet. Das Fenster für die globalen Variablen öffnen Sie über das Menü Fenster|Global.

Variablen-Fenster
Die Fenster für globale und modulspezifische Variablen sind weitgehend identisch aufgebaut. Links erscheint eine Liste aller Variablen, die darin ausgewählte Variable wird im Kopf des Fensters angezeigt, in der rechten Liste erscheint ein Dump der zugehörigen Daten. Sie können

im Exe-Fenster eine Token-Zeile doppelt anklicken, dann wird das Fenster mit den Modul-Variablen geöffnet und gleich noch die im Token verwendete Variable ausgewählt. In der Liste der Variablen finden Sie den Offset im Datenbereich, den gespeicherten Wert, die laufende Nummer des Unterprogramms, in dem die Variable lokal definiert ist, und den Namen der Variablen. Ganz am Anfang der Liste liegen die Namen aller Functions eines Moduls, im globalen Bereich können zusätzlich die Type-Deklarationen des Programms vorkommen.

Vorläufig werden die Namen der Variablen nicht aus dem Quelltext übernommen, Sie können jedoch allen Variablen die richtigen Namen zuweisen. Hierfür wählen Sie die Variable in der Liste aus, tragen den Namen ein und weisen diesen mit dem Button **Name** der Variablen zu.

Die Zuweisung von Datentypen ist etwas komplizierter, dazu wählen Sie den Scope (global, modul, parameter, lokal...) und den Typ der Variablen aus und weisen diesen der ausgewählten Variablen mit dem Button **As** zu. Bei Feldern, festen Strings, Types und Objekten wird der Typ automatisch ermittelt, sofern der von Ihnen vorgegebene Typ korrekt ist. Im Dump werden alle Angaben zu den Variablen angezeigt, daher können dort auch zu einfachen Variablen mehrere Worte erscheinen. Kleinere feste Felder werden oft direkt abgespeichert, so daß mit einem geeigneten Tool eine feste Initialisierung im compilierten Programm möglich ist!

Undefinierte Variablentypen stammen hauptsächlich von Variablen und Konstanten, die im Programm nicht verwendet werden und der Discompiler daher die Lage und die Typen der Variablen nicht ermitteln konnte. Sie können alle zugehörigen Deklarationen aus Ihrem Programm streichen, ebenso alle nicht verwendeten Parameter oder ganze Unterprogramme, die an unvollständigen oder gar fehlerhaften Parameter-Typen erkennbar sind. Nicht verwendete Parameter und Funktions-Ergebnisse sind auch am Offset 0000 im Dump erkennbar.

Globale Variablen und Konstanten werden in den Modulen durch die Offsets beschrieben, unter denen die Variablen im globalen Datenbereich abgelegt sind. Bei Modul-Konstanten wird der Wert im Modul gespeichert, während bei echten Modul-Variablen hier immer der Wert Null eingetragen ist. Konstante Strings werden durch Pointer in einen eigenen Speicherbereich beschrieben, während die Werte von numerischen Konstanten direkt gespeichert sind.

Speicherung
In der Lite-Version werden alle Namen und Typen gespeichert und sind beim nächsten Aufruf von VBMDis wieder verfügbar. Dazu geben Sie ein eigenes Verzeichnis für das Projekt vor, in dem die verschiedenen Dateien abgelegt werden. Wenn Sie das Programm neu compilieren, sollten Sie alle Dateien in diesem Verzeichnis oder das ganze Verzeichnis löschen, da der Discompiler sonst über die vorgenommenen Änderungen stolpern wird.

Custom-Controls
In der Lite-Version ist die Erfassung und Speicherung der Beschreibungen von Custom-Controls (VBX-Dateien) mit dem VBX-Tool **VBCtrl** möglich. Dieses Programm wurde inzwischen weitgehend überarbeitet, die bisherige provisorische Version wird nicht mehr benötigt und auch nicht mehr unterstützt.

Der Discompiler meldet sich jedesmal mit einer MsgBox, wenn er auf ein unbekanntes Steuerelement stößt. Lassen Sie diese MsgBox offen und starten Sie **VBCtrl**. Wählen Sie dort die gesuchte Datei aus und öffnen Sie diese mit einem Doppelklick oder über das Menü. Daraufhin erscheint ein Fenster mit allen globalen Entries der Datei, in dem Sie die Kontroll-Prozeduren der Steuerelemente suchen müssen. Diese Prozeduren sind meistens am Namen **...CTRLPROC** erkennbar (brav vom CDK abgeschrieben). Wenn Sie einen Namen aus der linken Liste auswählen, erscheinen in der mittleren Liste alle Verweise auf diese Prozedur, die mögliche Definitionen des Controls sind. In der rechten Liste wird eine kurze Beschreibung der vermuteten Definition angezeigt, die sie in der mittleren Liste ausgewählt haben. Wenn dabei

Fehlermeldungen ausgegeben werden, haben Sie mit hoher Wahrscheinlichkeit keine Beschreibung getroffen. Achten Sie auf die Version und den Klassennamen des Controls! Bei gleichen Klassennamen (CN) wählen Sie die Beschreibung mit der höchsten Versionsnummer (VN), möglichst 0300 für Visual Basic 3.00. Mit dem Button oberhalb der Beschreibung wird die komplette Beschreibung aus der Datei geladen und in einem weiteren Fenster angezeigt. Schließen Sie dieses Fenster wieder und laden Sie gleichermaßen alle Controls, die sich in der VBX-Datei befinden.

Wenn Sie das Verweis-Fenster schließen, werden die gefundenen Beschreibungen gespeichert, in der Demo-Version erscheint hier allerdings nur eine Fehlermeldung die Sie zum Anlaß nehmen sollten, sich für die vollständigen Versionen der Programme registrieren zu lassen (siehe Register.wri).

Nun wählen Sie in der MsgBox von **VBMDis** den Button **Wiederholen**, und das Programm sollte dann die neu erstellte Beschreibung finden und verwenden. Dazu sollten **VBCtrl** und **VBMDis** im gleichen Verzeichnis liegen, andernfalls müssen Sie die Beschreibungen (*.300) in das Verzeichnis von **VBMDis** kopieren. Die Speicherung der Beschreibungen wurde gegenüber der bisherigen Version stark komprimiert, und nur **VBMDis** ist in der Lage, die von Ihnen erstellten Beschreibungen zu verwenden. Die mitgelieferten Beschreibungen im alten Format sind wesentlich umfangreicher und werden sowohl von **VBMDis** als auch von **VBDis** benötigt, Sie sollten diese Dateien keinesfalls löschen oder verändern.

Ausblick

Da die Funktionen von **VBMDis** zur Optimierung von Programmen verwendet werden können, ist eine Trennung von **DoDi's VB-Tools** in Discompiler und Entwickler-Werkzeuge nicht mehr sinnvoll. Die Registrierung wurde daher so abgeändert, daß nunmehr nur noch VB-Tools mit oder ohne Discompiler ausgeliefert werden. Die bisher ausschließlich für den Discompiler registrierten Anwender erhalten weiterhin die Updates zu VBMDis und VBCtrl, sollten jedoch ein Upgrade auf die kompletten Tools ins Auge fassen, damit sie ihre Programme mit **VBPro** gegen Discompiler schützen können.

Bitte beachten Sie auch, daß **VBMDis3** wie auch **VBDis3** in der Lite-Version keine Module mit mehr als **64KB Code** verarbeiten können. Wenn sie eine entsprechende Fehlermeldung erhalten, teilen Sie die zu großen Module in mehrere kleine Module auf. Da VB 4.0 vor der Türe steht, bleibt mir vorläufig keine Zeit, die leider sehr komplexen Änderungen vorzunehmen, um die Discompiler für VB 3.0 auf so große Programme umzustellen. Der Discompiler für VB 4.0 wird keine derartige Einschränkung mehr aufweisen.