



Münchener Str. 4a
D-82131 Gauting / Germany
Tel. +49-89-8931043/45
E-Mail: contact@crst.de
Web: www.crst.de

AP-Note

Erfolgreiche Migration von VB6 nach VB.Net®

Thomas Criegee CRST GmbH

Einleitung

In vielen Firmen sind noch Visual Basic® (VB6) Programme, seinerzeit erstellt mit Hilfe der Entwicklungsumgebung Visual Studio® 6.0 von Microsoft, in Verwendung. Sie werden gepflegt, mit Updates versorgt und teilweise noch weiterentwickelt.

Viele Bücher und Dokumente beschäftigen sich mit dem Thema Migration/Upgrade von VB6 nach VB.Net. Wie groß der Aufwand der Migration auf die moderne Dot.Net-Technologie VB.NET in der Praxis ist, welche Risiken dabei auftreten können, welche Kosten entstehen und wo der eigentliche Vorteil (oder auch Nachteil) für den Anwender liegt, wird meistens nicht so deutlich erwähnt.

Die folgende AP-Note beschreibt (unter Verwendung von Visual Studio® 2005, kurz VS2005) die einzelnen Schritte der Migration beispielhaft für ein kleineres VB6 Projekt, bestehend aus ca. 25 einzelnen Formen/Modulen/Klassen. Diese Vorgehensweise wurde auch für umfangreiche VB6 Projekte, bestehend aus mehr als 100 Formen/Modulen/Klassen, eingesetzt. Letzendlich wird eine grobe Zeitabschätzung für die Migration umfangreicher Projekte gegeben, ausserdem werden weitere Themen, z.B. Fehlerbehandlung (Run Time Errors) und Verschlüsselung der Net Assemblies besprochen.

Schritt 1

Konvertieren des VB6 Projektes mit dem Visual Basic 2005 Aktualisierungs-Assistenten

Vor Beginn der Migration sollten unbedingt die neuesten VS2005 Service Packs installiert werden, ansonsten können bei der Konvertierung Probleme mit dem VS2005-Aktualisierungs-Assistenten auftreten.

In diesem Beispiel wurde das VB6 Projekt *NetMigration.vbp* mit dem Visual Basic Aktualisierungs-Assistent konvertiert, es entstand daraus das neue VB.Net Projekt *NetMigration.vbproj*.

Schritt 2

Umbenennen der vom Aktualisierungs-Assistenten erzeugten Klassen/Formen/Module

Da bei VB.Net alle Filenamen standardmäßig die Erweiterung *.vb tragen (bei VB6 Module: *.bas, Formen: *.frm, Klassen: *.cls etc.), vergibt der Aktualisierungs-Assistent teilweise neue Filenamen zur Unterscheidung von Formen, Klassen und Modulen (teilweise mit Hinzufügen von „_bas“ oder „_frm“ im Dateinamen selber).

Aufgrund der Übersichtlichkeit wurden als erstes die vom Aktualisierungs-Assistent generierten Klassen/Formen/Module nach einer einheitlichen Namenskonvention umbenannt:

Klassen: vbcXXXX.vb

Formen: vbfYYYY.vb

Module: vbmZZZ.vb

Damit ist innerhalb des Projektmappen-Explorers eine saubere Trennung sowie ein schnelles Auffinden der einzelnen Softwaremodule gewährleistet.

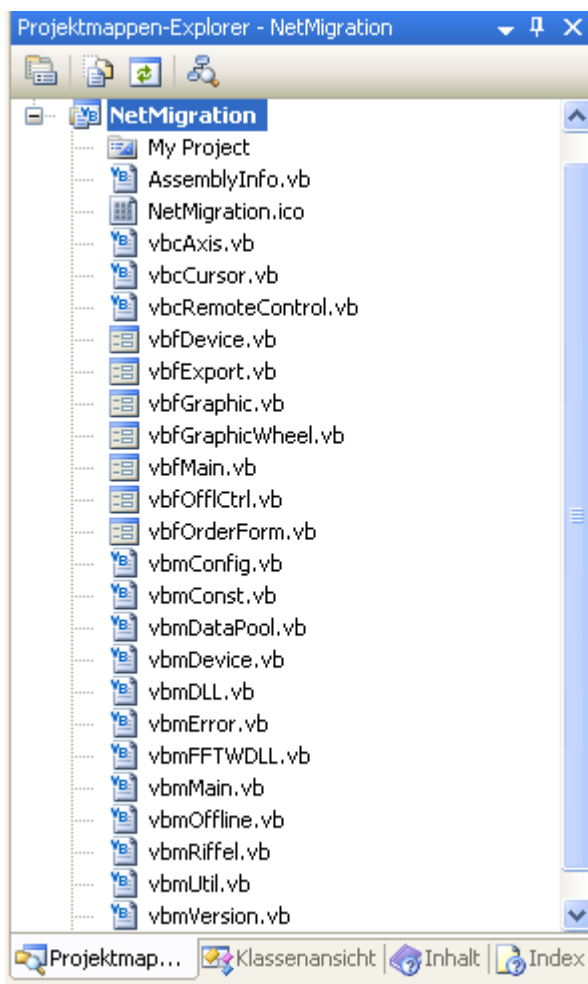


Bild 1: Projektmappen-Explorer mit einheitlicher Namenskonvention

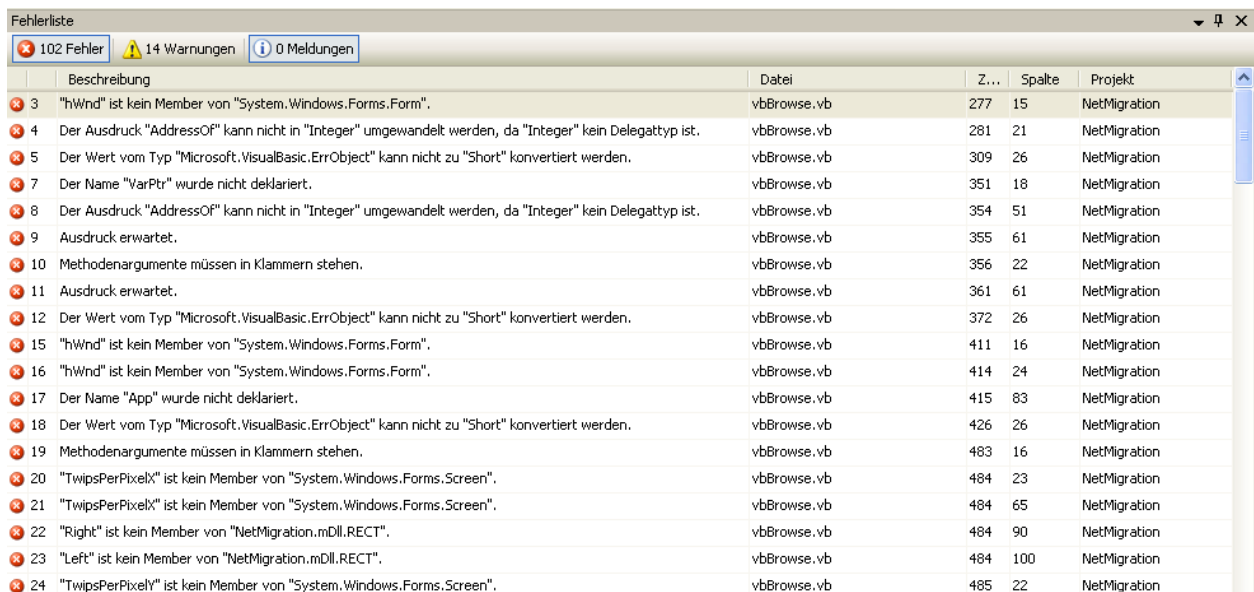
Schritt 3

Entfernen der syntaktischen Fehler im Code

Die durch die Konvertierung mit dem Aktualisierungs-Assistent generierten Klassen/Formen/Module weisen aufgrund der Inkompatibilität der Sprachen VB6 und VB.Net mehr oder weniger sog. EWIs auf, also Fehler (Errors), „Upgrade Warnings“ und andere „Issues“, wobei die Betonung deutlich auf „mehr“ liegt. Visual Studio zeigt Fehler und Warnungen in einem extra Fenster an, die „Upgrade Warnings“ werden zeilenweise in den konvertierten Code eingefügt, mit Hilfe der STRG-Taste und Klick auf den in der Zeile enthaltenen Link wird die dazugehörige Online-Hilfe angezeigt.

Die angezeigten Fehler resultieren unter anderem aus:

1. Inkompatibilität der Sprachen VB6 und VB.Net
2. Strengerer Objekt-Orientierung/Auslegung von VB.Net
3. Trennung von verwaltetem und nicht-verwaltetem Speicher
4. Änderung der Form-Eigenschaften (Graphik-Auflösung etc.)
5. Änderung vorhandener Objekte (Application Object, Dialoge, Menüs, o.ä.)



	Beschreibung	Datei	Z...	Spalte	Projekt
3	"hWnd" ist kein Member von "System.Windows.Forms.Form".	vbBrowse.vb	277	15	NetMigration
4	Der Ausdruck "AddressOf" kann nicht in "Integer" umgewandelt werden, da "Integer" kein Delegates Typ ist.	vbBrowse.vb	281	21	NetMigration
5	Der Wert vom Typ "Microsoft.VisualBasic.ErrObject" kann nicht zu "Short" konvertiert werden.	vbBrowse.vb	309	26	NetMigration
7	Der Name "VarPtr" wurde nicht deklariert.	vbBrowse.vb	351	18	NetMigration
8	Der Ausdruck "AddressOf" kann nicht in "Integer" umgewandelt werden, da "Integer" kein Delegates Typ ist.	vbBrowse.vb	354	51	NetMigration
9	Ausdruck erwartet.	vbBrowse.vb	355	61	NetMigration
10	Methodenargumente müssen in Klammern stehen.	vbBrowse.vb	356	22	NetMigration
11	Ausdruck erwartet.	vbBrowse.vb	361	61	NetMigration
12	Der Wert vom Typ "Microsoft.VisualBasic.ErrObject" kann nicht zu "Short" konvertiert werden.	vbBrowse.vb	372	26	NetMigration
15	"hWnd" ist kein Member von "System.Windows.Forms.Form".	vbBrowse.vb	411	16	NetMigration
16	"hWnd" ist kein Member von "System.Windows.Forms.Form".	vbBrowse.vb	414	24	NetMigration
17	Der Name "App" wurde nicht deklariert.	vbBrowse.vb	415	83	NetMigration
18	Der Wert vom Typ "Microsoft.VisualBasic.ErrObject" kann nicht zu "Short" konvertiert werden.	vbBrowse.vb	426	26	NetMigration
19	Methodenargumente müssen in Klammern stehen.	vbBrowse.vb	483	16	NetMigration
20	"TwipsPerPixelX" ist kein Member von "System.Windows.Forms.Screen".	vbBrowse.vb	484	23	NetMigration
21	"TwipsPerPixelX" ist kein Member von "System.Windows.Forms.Screen".	vbBrowse.vb	484	65	NetMigration
22	"Right" ist kein Member von "NetMigration.mDI.RECT".	vbBrowse.vb	484	90	NetMigration
23	"Left" ist kein Member von "NetMigration.mDI.RECT".	vbBrowse.vb	484	100	NetMigration
24	"TwipsPerPixelY" ist kein Member von "System.Windows.Forms.Screen".	vbBrowse.vb	485	22	NetMigration

Bild 2: Syntaktische Fehler

Bild 3: Upgrade Warnings

Die angezeigten Fehler/Warnungen müssen nun für Module, Klassen und Formen sukzessive eliminiert werden. Dies ist bzw. kann sehr aufwändig sein, einige der notwendigsten Änderungen sind in der folgenden Tabelle (Bild 4) punktuell aufgeführt.

Des weiteren empfiehlt sich dabei gleich auch eine (hoffentlich vorhandene) Fehlerbehandlung, bestehend z.B. aus „On Error Goto“, durch die neuen .NET Try/Catch Konstruktionen im Code zu ersetzen.

Nr.	Fehler aufgrund Aktualisierung	Migration nach .NET (Bemerkung/Beispiel)
1.	Ersetzen einfacher syntaktischer Fehler	Methodenargumente in Klammern Überprüfen ByVal, ByRef
2.	Überprüfen verwendeter Arrays	In .NET nicht mehr möglich: Verkleinerung von Arrays mit ReDim Fixed Length Arrays, Negative Array Indizes Option Base 1 (VB.Net Lower Bound immer 0) In .NET neu: Explizite Angabe der Dimension(en) eines Arrays Ersetzen von Funktionen Array.Copy statt CopyMemory Array.Clear statt ZeroMemory In .NET umständlich: Strukturen (UDTs) mit darin enthaltenen Arrays müssen initialisiert werden.
3.	Ersetzen nicht mehr vorhandener Funktionen, Arrays	In .NET nicht mehr vorhanden: Variant, LenB, Hook-Funktionen
4.	Marshaling für Funktionen die auf den nicht verwalteten Speicher zugreifen (VarPtr, ObjPtr, StrPtr, AddressOf)	Speicherverwaltung mit Marshaling .NET: System.Runtime.InteropServices.Marshal
5.	Marshaling für DLL Funktionen die auf den nicht verwalteten Speicher zugreifen	Keine Parameterübergabe mit „As Any“ mehr Anlegen/Verwenden von verwaltetem Speicher (Marshal.AllocHGlobal ...) .NET: System.Runtime.InteropServices.Marshal
6.	Ersetzen der Menüs / Popup Menüs	.NET: System.Windows.Forms.ContextMenuStrip
7.	Ersetzen der Farbattribute	.NET: System.Drawing.Color
8.	Ersetzen des App Objektes	.NET: My.Application
9.	Ersetzen des Screen Objektes	.NET: System.Windows.Forms.Screen System.Windows.Forms.Cursor
10.	Überprüfen verwendeter Datentypen	Bsp.: Short, Integer, Long
11.	Einbau nicht verwendeter bzw. geänderter Attribute von Steuerelementen	Bsp.: TextBox.Text = „123“ statt TextBox = „123“
12.	Ersetzen allgemeiner Steuerelemente (Controls) durch explizite Angabe bzw. Verwendung von CType, GetType o.ä.	.NET: Explizite Angabe des Steuerelemente Typs
13.	Ersetzen von Steuerelemente-Arrays	.NET: Instanz statt Array
14.	Ersetzen der Dialoge (CommonDialog)	.NET: OpenFileDialog, SaveFileDialog, FontDialog, PrintDialog, ColorDialog
15.	Ersetzen der Online-Hilfe (HelpContextIDs)	.NET: Form_HelpRequested()
16.	Enumerieren von Steuerelementen in Formen, Rahmen (Frames) etc.	.NET: Rekursive statt flache Hierarchie Explizite Verwendung von TypeOf

Bild 4: Fehler durch Aktualisierung

Schritt 4

Entfernen der laufzeitabhängigen Fehler

Nachdem alle syntaktischen Fehler und Warnungen elimiert sind, kann mit dem Debugger die Suche nach laufzeitbedingten Fehlern im Programmablauf fortgesetzt werden.



Bild 5: Keine Fehler/Warnungen mehr

Zum Entfernen der Fehler beim Programmablauf wird das Programm mit dem in Visual Studio enthaltenen Debugger gestartet. Dabei werden u.a. folgende Fehler aufgespürt:

1. Indexüberschreitungen von Arrays
2. Fehlerhafte DLL Aufrufe
3. Fehler beim Zugriff auf verwalteten/nichtverwalteten Speicher
4. Nicht vorhandene/fehlerhafte Attribute von Objekten
5. Nicht vorhandene/fehlerhafte Attribute von Steuerelementen

Schritt 5

Entfernen aller noch vorhandenen VB6-Verweise und Ersetzen durch VB.Net Elemente

Ein weiterer Schritt der Migration kann das Entfernen der Verweise auf (noch) im Projekt vorhandene VB6-Steuerelemente (ActiveX Controls) sowie Arrays von Steuerelementen und Ersetzen dieser durch Windows.Forms Elemente darstellen. Dies hat den Vorteil, dass sämtliche ActiveX Controls aus der Installation entfernt werden können und somit auch auf dem Rechner des Kunden nicht mehr installiert und nicht mehr registriert werden müssen.

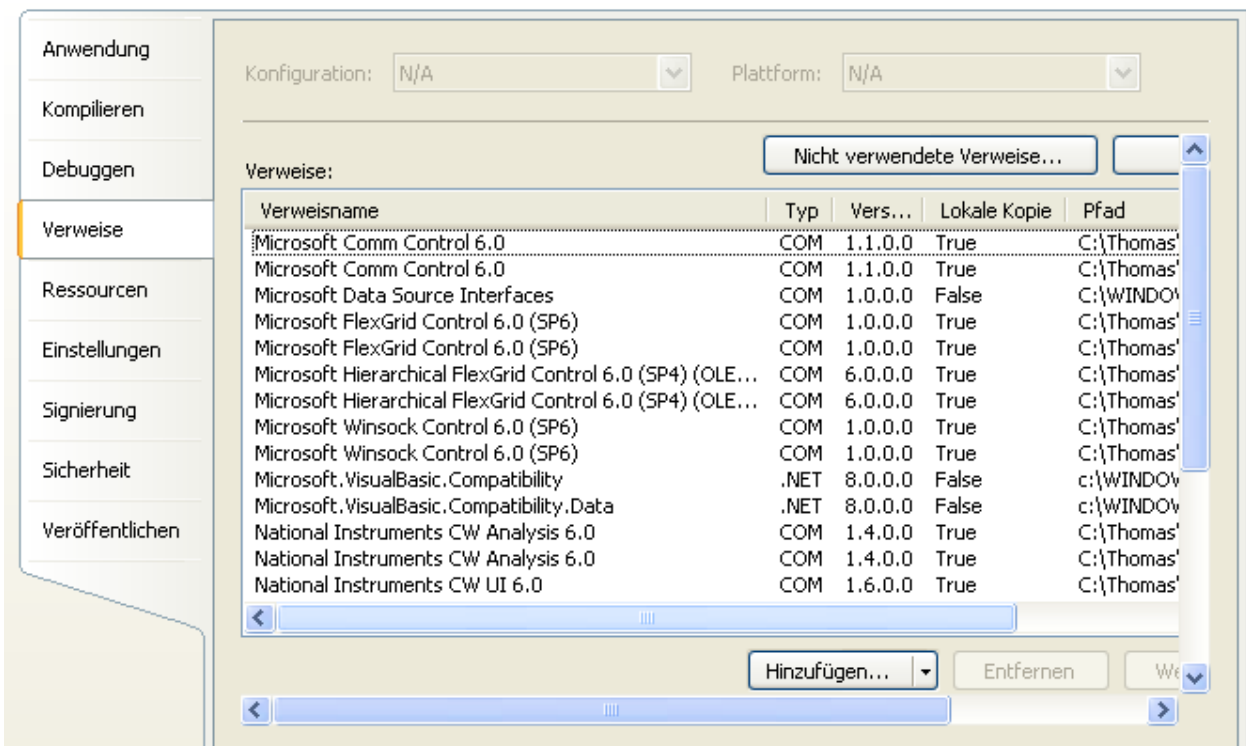


Bild 6: Vorhandene Verweise

Dieser Schritt ist ziemlich aufwändig, insbesondere das Ersetzen vorhandener Steuerlemente-Arrays, da diese sowohl im Designer als auch im dazugehörigen Code sukzessive entfernt und durch die entsprechenden Windows.Forms Elemente mit den dazugehörigen Event-Prozeduren ersetzt werden müssen.

Für dieses Migration werden u.a. folgende Substitutionen vorgenommen

VB6 ActiveX Control (alt)	VB.Net Element bzw. Klasse (neu)
Frame	System.Windows.Forms.GroupBox
MSFlexGrid, MSHFlexGrid	System.Windows.Forms.DataGridView
CommControl	System.IO.Ports.SerialPort
WinSock Control, Internet Transfer Control	System.Net
etc.	

Schritt 6

Nachbearbeiten des Projektes, Fehlerbehandlung, Verschleierung/Verschlüsselung

Die .NET Klassenbibliothek stellt für die Behandlung von Laufzeitfehlern (z.B. in Try/Catch Konstrukten) die Exception-Klasse sowie StackTrace und StackFrame Funktionen bereit. Deren Mitglieder erlauben eine Anzeige der zuletzt aufgerufenen Funktionen plus die Zeilennummern der Funktionsaufrufe (allerdings nicht die Zeilennummer, in der der Fehler aufgetreten ist !!) in der VS2005 Debug-Version. In der Release-Version sind allerdings weder Funktionsname noch Zeilennummer verfügbar.

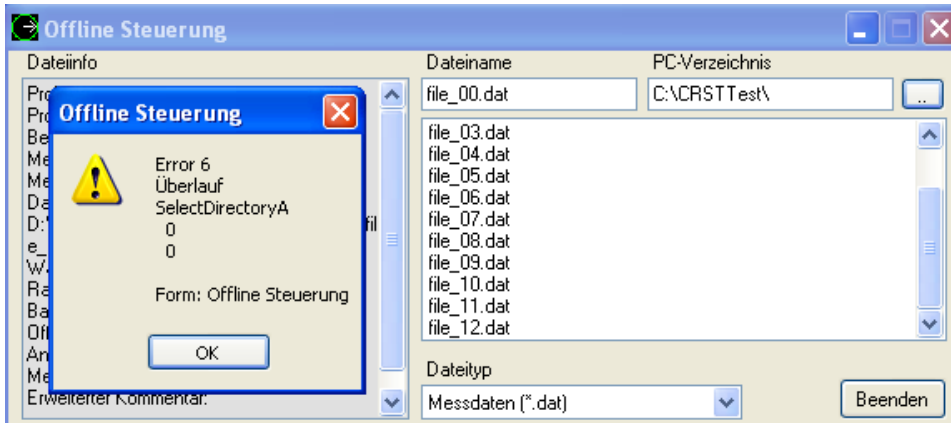


Bild 7: Keine Informationen über Laufzeitfehler in der VS2005 Release Version

Um im Fehlerfall detaillierte Informationen zu erhalten, muss der gesamte Quellcode des Projektes nachbearbeitet werden. Hierfür wurde bei CRST das Tool VBLines entwickelt, das den vorhandenen Quellcode so bearbeitet, dass im Fehlerfall die aktuelle Zeilennummer der Anweisung des Moduls, in dem der Fehler aufgetreten ist, zur Verfügung steht.

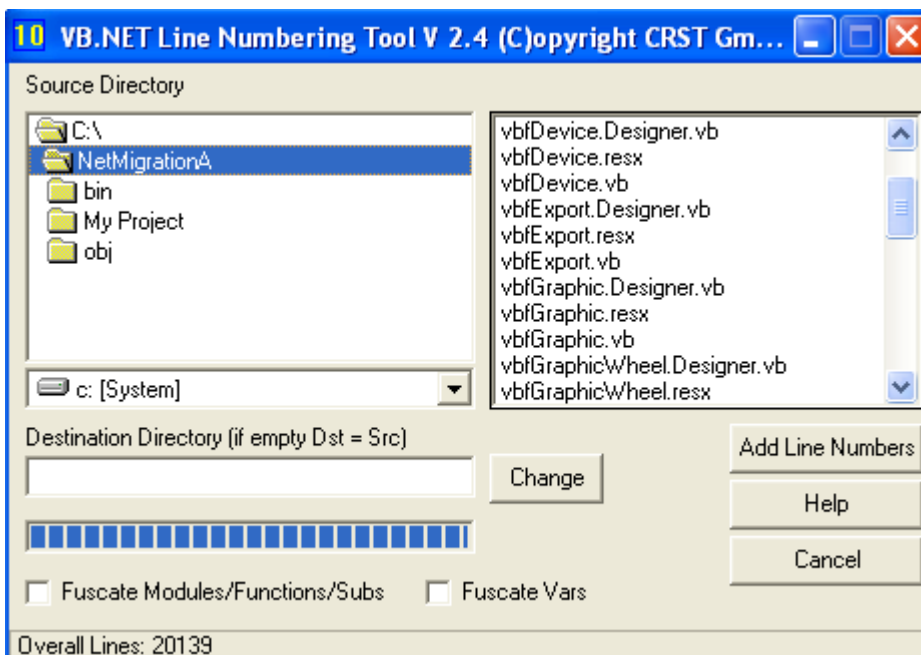


Bild 8: Das CRST Tool VBLines.

Durch diese Nachbehandlung ist es möglich, die Fehlernummer und -Ursache, die Zeilennummer, die Funktion/Klasse und den dazugehörigen Modul von Laufzeitfehlern exakt anzuzeigen.

Dies ist insbesondere für Service und Hotline eine unabdingbare Voraussetzung, um Kunden bei Problemen jederzeit schnell mit Workarounds, Bugfixes etc. helfen zu können.

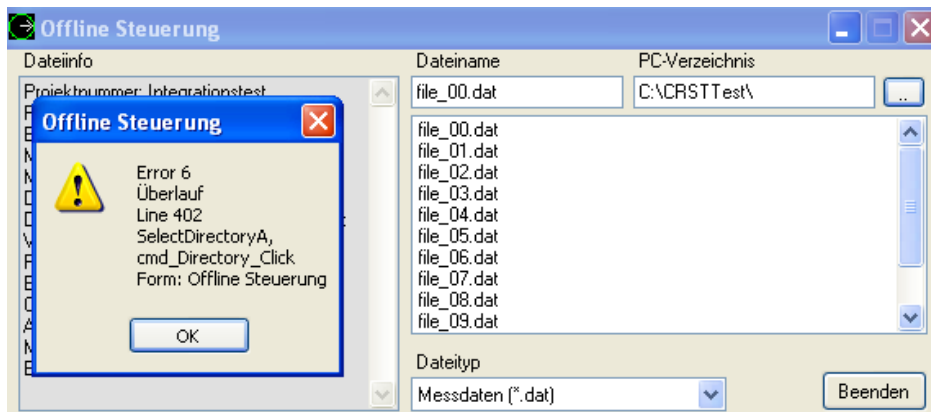


Bild 9: Genaue Zeilen-Information über Laufzeitfehler nach Bearbeiten mit VBLines.

Ein weiterer nicht zu unterschätzender Punkt ist der folgende: .Net Anwendungen bzw. Assemblies bestehen aus interpretierbarem Quelltext!

Ohne Verschleierung („Obfuskation“) oder Verschlüsselung der Net Assemblies kann der Quelltext mit geeigneten Tools angesehen oder verändert werden.

Deswegen sollten sämtliche Softwaremodule des Projektes mindestens verschleiert („obfuscated /dotfuscated“), besser noch, mit geeigneten Drittanbieter-Tools verschlüsselt werden. Diese Tools verschlüsseln den Code vor der Auslieferung zum Kunden und entschlüsseln ihn erst zur Laufzeit wieder.

Schritt 7

Erzeugen der Installation

Als letztes muss das Projekt mit dem nachbearbeiteten Source Code kompiliert und in ein lauffähiges *.exe File verwandelt werden.

Zusammen mit allen für das Projekt nötigen Dateien wird dann mit einem geeigneten Tool (z.B. Inno Setup) die Installation bzw. die Setup-Datei erzeugt.

Schritt 8

Wiederholen des gesamten Integrations- und Systemtestes

Als letzter Schritt muss der gesamte Integrations- und Systemtest anhand der zugrundeliegenden Testpläne wiederholt werden.

Dazu gehört auch der Test der in Schritt 7 erzeugten Installation auf einem „nackten“ Rechner, also auf einem Rechner, der ausser dem Betriebssystem keinerlei andere installierte Programme enthält.

Zusammenfassung

Die anfangs gestellte Frage, wie groß der Aufwand der Migration auf die moderne Dot.Net-Technologie VB.NET ist, welche Risiken dabei auftreten können, welche Kosten entstehen und wo der eigentliche Vorteil (oder auch Nachteil) für den Anwender liegt, kann abschliessend in etwa folgendermaßen beurteilt werden:

- Bei kleineren Projekten (ca. 20-30 Programmmodule), wenigen VB6 Standard-Steuerelementen und einigermaßen „sauberer“ Programmierung ohne Tricks (geringer Einsatz von direkten Windows API-Calls) ist der Migrationsaufwand für erfahrene Softwareentwickler und vorhandener Projekt-Infrastruktur überschaubar (2- 4 Mann/Frau-Wochen).
- Bei größeren Projekten (ab ca. 100 Programmmodule) mit vielen VB6 Standard-Steuerelementen, Steuerelementen von Drittanbietern sowie vielen verwendeten Windows API-Calls und anderen DLL-Aufrufen ist der Migrationsaufwand auch für erfahrene Softwareentwickler beträchtlich (mehrere Mann/Frau-Monate).

Nicht zu unterschätzen sind letztendlich auch die der Codesicherheit dienenden Maßnahmen, die erneut durchzuführenden Software-Tests sowie die abschliessende Verschleierung bzw. Verschlüsselung des Codes vor der Auslieferung.