

EINSPRUNGUNKT DLL DLLS MIT VB

1. Übersicht

In diesem Tutorial wird erläutert, wie man ohne die Hilfe von teuren AddIns Einsprungpunkt-DLLs mit Visual Basic erstellen kann. Zusätzlich wird kurz erläutert, welche versteckten Möglichkeiten sich noch bieten. Außerdem enthält dieses Tutorial zusätzlich ein Beispiel für eine DLL, die zwei Funktionen exportiert und ein Beispielpjekt, das die Verwendung dieser DLL veranschaulicht.

Mit freundlichen Grüßen

Tobias Koch TobiKochFN@T-Online.de

2. Allgemeines

Bevor wir WinAPI-DLLs erstellen können, müssen wir verstehen, wie der Visual Basic Compiler funktioniert bzw. wie man in anderen Programmiersprachen (z.B. C++) solche Dateien erzeugt.

Wenn Sie im Menü des VB Editors "Kompilieren" wählen, übersetzt der Editor den von Ihnen getippten Code in eine Zwischensprache. Diese Dateien werden an einen zweiten Compiler übergeben (c2.exe), der daraus Objektmodule (*.obj) erstellt. Diese wiederum werden vom Linker (link.exe) zusammengefügt, d.h. eine ausführbare Datei (*.exe, *.dll, *.ocx) wird erstellt.

Wenn wir eine WinAPI-DLL erstellen möchten, müssen wir also ein eigenes Programm zwischen c2.exe und link.exe schalten, das die dem Linker übergebenen Befehlszeilenparameter entsprechend abändert.

3. Grundlagen

Wenn wir den Linker mit einem eigenen Programm ersetzen, das die übergebenen Befehlszeilenparameter anzeigt, können wir verstehen, wie der Linker aus mehreren Objektmodulen ein Programm erzeugt:

```
"C:\Test\Module11.OBJ" "C:\Test\Test1.OBJ" "C:\Programme\Visual
Basic\VBAEXE5.LIB" /ENTRY:__vbaS /OUT:"C:\Test\Test.exe" /BASE:0x400000
/SUBSYSTEM:WINDOWS,4.0 /VERSION:1.0 /INCREMENTAL:NO /OPT:REF
/MERGE:.rdata=.text /IGNORE:4078
```

Nun betrachten wir die Befehlszeilenparameter des Linkers:

```
Microsoft (R) 32-Bit Incremental Linker Version 4.20.6164
Copyright (C) Microsoft Corp 1992-1996. All rights reserved.
```

```
usage: LINK [options] [files] [@commandfile]
```

```
options:
```

```
    /ALIGN:#
    /BASE:{address|@filename,key}
    /COMMENT:comment
    /DEBUG
    /DEBUGTYPE:{CV|COFF|BOTH}
    /DEF:filename
    /DEFAULTLIB:library
```

```

/DLL
/DRIVER[:UPONLY]
/ENTRY:symbol
/EXETYPE:DYNAMIC
/EXPORT:symbol
/FIXED
/FORCE[:{MULTIPLE|UNRESOLVED}]
/GPSIZE:#
/HEAP:reserve[,commit]
/IMPORT:[symbol][,][LIB=container][,WEAK=1]
/IMPORT:[CURRENTVER=#][,][OLDCODEVER=#][,][OLDAPIVER=#]
/IMPLIB:filename
/INCLUDE:symbol
/INCREMENTAL:{YES|NO}
/LIBPATH:path
/MAC:{BUNDLE|NOBUNDLE|TYPE=xxxx|CREATOR=xxxx|INIT=symbol|TERM=symbol}
/MAC:{MFILEPAD|NOMFILEPAD}
/MACDATA:filename
/MACHINE:{IX86|MIPS|ALPHA|PPC|M68K|MPPC}
/MACRES:filename
/MAP[:filename]
/MERGE:from=to
/NODEFAULTLIB[:library]
/NOENTRY
/NOLOGO
/OPT:{REF|NOREF}
/ORDER:@filename
/OUT:filename
/PDB:{filename|NONE}
/PROFILE
/RELEASE
/SECTION:name,[E][R][W][S][D][K][L][P][X]
/SHARED
/STACK:reserve[,commit]
/STUB:filename
/SUBSYSTEM:{NATIVE|WINDOWS|CONSOLE|POSIX}[,#[.##]]
/SWAPRUN:{NET|CD}
/VERBOSE[:LIB]
/VERSION:#[.##]
/VXD
/WARN[:warninglevel]
/WS:AGGRESSIVE

```

Auf die Bedeutung dieser Parameter möchte ich nicht näher eingehen, hier finden Sie eine genaue Beschreibung:

[MSDN Linker Reference](http://msdn.microsoft.com/library/default.asp?url=/library/en-us/vccore98/HTML/_core_linker_reference.asp) (http://msdn.microsoft.com/library/default.asp?url=/library/en-us/vccore98/HTML/_core_linker_reference.asp)

Die Parameter, die uns interessieren, sind "/DLL" und "/DEF:FileName". Durch "/DLL" wird angegeben, dass eine Bibliothek erstellt werden soll, durch die zweite Option können wir eine Exportdefinitionsdatei angeben, mit deren Hilfe wir bestimmen können, welche Funktionen exportiert werden sollen.

4. Linker-Controller

Nun müssen wir ein Programm entwickeln, dass zwischen dem zweiten Compiler und dem Linker aufgerufen wird und somit die vom IDE übergebenen Befehlszeilenparameter abändert.

Erstellen Sie ein neues Projekt und fügen Sie diesem ein Formular (frmMain) und ein Modul (modFunctions) ein. In das Modul müssen Sie einen Teil des Codes des Tipps 148 einfügen:

```
'Dieser Source stammt von http://www.activevb.de
'und kann frei verwendet werden. Für eventuelle Schäden
'wird nicht gehaftet.

'Um Fehler oder Fragen zu klären, nutzen Sie bitte unser Forum.
'Ansonsten viel Spaß und Erfolg mit diesem Source!

'----- Anfang Projektdatei Project1.vbp -----
'----- Anfang Formular "Form1" alias Form1.frm -----

'Control CommandButton: Command1

Option Explicit

'Diverse APIs deklarieren
Private Declare Function CreateProcess Lib "Kernel32" Alias _
    "CreateProcessA" ( _
        ByVal lpAppName As Long, _
        ByVal lpCmdLine As String, _
        ByVal lpProcAttr As Long, _
        ByVal lpThreadAttr As Long, _
        ByVal lpInheritedHandle As Long, _
        ByVal lpCreationFlags As Long, _
        ByVal lpEnv As Long, _
        ByVal lpCurDir As Long, _
        lpStartupInfo As STARTUPINFO, _
        lpProcessInfo As PROCESS_INFORMATION _
    ) As Long

Private Declare Function WaitForSingleObject Lib "Kernel32" ( _
    ByVal hHandle As Long, _
    ByVal dwMilliseconds As Long _
) As Long

Private Declare Function CloseHandle Lib "Kernel32" ( _
    ByVal hObject As Long _
) As Long

'Einige Konstanten benennen
Private Const NORMAL_PRIORITY_CLASS = &H20&
Private Const INFINITE = -1&

'Einige Datentypen erstellen
Private Type STARTUPINFO
    cb As Long
    lpReserved As String
    lpDesktop As String
    lpTitle As String
    dwX As Long
    dwY As Long
    dwXSize As Long
    dwYSize As Long
    dwXCountChars As Long
```

```

dwYCountChars As Long
dwFillAttribute As Long
dwFlags As Long
wShowWindow As Integer
cbReserved2 As Integer
lpReserved2 As Integer
hStdInput As Long
hStdOutput As Long
hStdError As Long
End Type

Private Type PROCESS_INFORMATION
    hProcess As Long
    hThread As Long
    dwProcessID As Long
    dwThreadId As Long
End Type

Public Function ShellWait(cmdline As String, Optional ByVal _
    bShowApp As Boolean = False) As Boolean
    'Diese Funktion führt einen Befehl (in CmdLine) aus.
    'Dabei wird das sich öffnende Fenster unsichtbar gemacht.
    'Diese Funktion wird erst beendet, wenn der Befehl
    'vollständig abgearbeitet ist.

    'Speicher reservieren
    Dim uProc As PROCESS_INFORMATION
    Dim uStart As STARTUPINFO
    Dim lRetVal As Long

    'Die Datentypen initialisieren
    uStart.cb = Len(uStart)
    uStart.wShowWindow = Abs(bShowApp)
    uStart.dwFlags = 1

    'Fenster erzeugen
    lRetVal = CreateProcess(0&, cmdline, 0&, 0&, 1&, _
        NORMAL_PRIORITY_CLASS, 0&, 0&, _
        uStart, uProc)

    'Warten, bis Fenster beendet wurde
    lRetVal = WaitForSingleObject(uProc.hProcess, INFINITE)

    'Fenster schließen
    lRetVal = CloseHandle(uProc.hProcess)

    'Rückgabewert setzen
    ShellWait = (lRetVal <> 0)
End Function

```

Falls Sie VB 5 nutzen, müssen Sie in das Modul zusätzlich eine Replace-Funktion einfügen, da wir diese später benötigen:

```

Public Function Replace(ByRef Text As String, _
    ByRef sOld As String, ByRef sNew As String, _
    Optional ByVal Start As Long = 1, _
    Optional ByVal Count As Long = 2147483647, _

```

```

Optional ByVal Compare As VbCompareMethod = vbBinaryCompare _
) As String

If LenB(sOld) Then

    If Compare = vbBinaryCompare Then
        ReplaceBin Replace, Text, Text, _
            sOld, sNew, Start, Count
    Else
        ReplaceBin Replace, Text, LCase$(Text), _
            LCase$(sOld), sNew, Start, Count
    End If

Else 'Suchstring ist leer:
    Replace = Text
End If
End Function

Private Static Sub ReplaceBin(ByRef Result As String, _
    ByRef Text As String, ByRef Search As String, _
    ByRef sOld As String, ByRef sNew As String, _
    ByVal Start As Long, ByVal Count As Long _
)
    Dim TextLen As Long
    Dim OldLen As Long
    Dim NewLen As Long
    Dim ReadPos As Long
    Dim WritePos As Long
    Dim CopyLen As Long
    Dim Buffer As String
    Dim BufferLen As Long
    Dim BufferPosNew As Long
    Dim BufferPosNext As Long

    'Ersten Treffer bestimmen:
    If Start < 2 Then
        Start = InStrB(Search, sOld)
    Else
        Start = InStrB(Start + Start - 1, Search, sOld)
    End If
    If Start Then

        OldLen = LenB(sOld)
        NewLen = LenB(sNew)
        Select Case NewLen
        Case OldLen 'einfaches Überschreiben:

            Result = Text
            For Count = 1 To Count
                MidB$(Result, Start) = sNew
                Start = InStrB(Start + OldLen, Search, sOld)
                If Start = 0 Then Exit Sub
            Next Count
            Exit Sub

        Case Is < OldLen 'Ergebnis wird kürzer:

            'Buffer initialisieren:
            TextLen = LenB(Text)
            If TextLen > BufferLen Then

```

```

    Buffer = Text
    BufferLen = TextLen
End If

'Ersetzen:
ReadPos = 1
WritePos = 1
If NewLen Then

    'Einzufügenden Text beachten:
    For Count = 1 To Count
        CopyLen = Start - ReadPos
        If CopyLen Then
            BufferPosNew = WritePos + CopyLen
            MidB$(Buffer, WritePos) = MidB$(Text, ReadPos, CopyLen)
            MidB$(Buffer, BufferPosNew) = sNew
            WritePos = BufferPosNew + NewLen
        Else
            MidB$(Buffer, WritePos) = sNew
            WritePos = WritePos + NewLen
        End If
        ReadPos = Start + OldLen
        Start = InStrB(ReadPos, Search, sOld)
        If Start = 0 Then Exit For
    Next Count

Else

    'Einzufügenden Text ignorieren (weil leer):
    For Count = 1 To Count
        CopyLen = Start - ReadPos
        If CopyLen Then
            MidB$(Buffer, WritePos) = MidB$(Text, ReadPos, CopyLen)
            WritePos = WritePos + CopyLen
        End If
        ReadPos = Start + OldLen
        Start = InStrB(ReadPos, Search, sOld)
        If Start = 0 Then Exit For
    Next Count

End If

'Ergebnis zusammenbauen:
If ReadPos > TextLen Then
    Result = LeftB$(Buffer, WritePos - 1)
Else
    MidB$(Buffer, WritePos) = MidB$(Text, ReadPos)
    Result = LeftB$(Buffer, WritePos + LenB(Text) - ReadPos)
End If
Exit Sub

Case Else 'Ergebnis wird länger:

    'Buffer initialisieren:
    TextLen = LenB(Text)
    BufferPosNew = TextLen + NewLen
    If BufferPosNew > BufferLen Then
        Buffer = Space$(BufferPosNew)
        BufferLen = LenB(Buffer)
    End If

```

```

'Ersetzung:
ReadPos = 1
WritePos = 1
For Count = 1 To Count
    CopyLen = Start - ReadPos
    If CopyLen Then
        'Positionen berechnen:
        BufferPosNew = WritePos + CopyLen
        BufferPosNext = BufferPosNew + NewLen

        'Ggf. Buffer vergrößern:
        If BufferPosNext > BufferLen Then
            Buffer = Buffer & Space$(BufferPosNext)
            BufferLen = LenB(Buffer)
        End If

        'String "patchen":
        MidB$(Buffer, WritePos) = MidB$(Text, ReadPos, CopyLen)
        MidB$(Buffer, BufferPosNew) = sNew
    Else
        'Position bestimmen:
        BufferPosNext = WritePos + NewLen

        'Ggf. Buffer vergrößern:
        If BufferPosNext > BufferLen Then
            Buffer = Buffer & Space$(BufferPosNext)
            BufferLen = LenB(Buffer)
        End If

        'String "patchen":
        MidB$(Buffer, WritePos) = sNew
    End If
    WritePos = BufferPosNext
    ReadPos = Start + OldLen
    Start = InStrB(ReadPos, Search, sOld)
    If Start = 0 Then Exit For
Next Count

'Ergebnis zusammenbauen:
If ReadPos > TextLen Then
    Result = LeftB$(Buffer, WritePos - 1)
Else
    BufferPosNext = WritePos + TextLen - ReadPos
    If BufferPosNext < BufferLen Then
        MidB$(Buffer, WritePos) = MidB$(Text, ReadPos)
        Result = LeftB$(Buffer, BufferPosNext)
    Else
        Result = LeftB$(Buffer, WritePos - 1) & MidB$(Text, ReadPos)
    End If
End If
Exit Sub

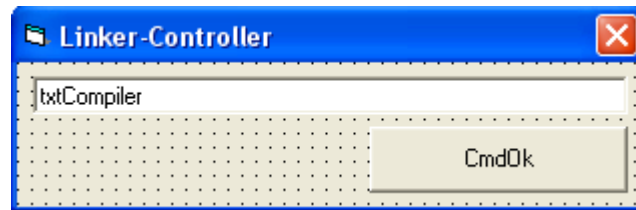
End Select

Else 'Kein Treffer:
    Result = Text
End If
End Sub

```

(Code von <http://www.vb-tec.de/replace.htm>)

Positionieren Sie auf dem Formular nun eine Textbox (txtCompiler) und einen Button (CmdOk) und fügen Sie den folgenden Code ein:



```
Private Sub CmdOk_Click()  
  
    Dim CmdLine As String  
    CmdLine = Command()  
  
    CmdLine = Replace(CmdLine, "/ENTRY:__vbaS", "/ENTRY:DLLMain")  
    CmdLine = Replace(CmdLine, "/BASE:0x400000", "/BASE:0x10000000")  
    CmdLine = CmdLine + " /DLL " + "/DEF:" + """" + txtCompiler.Text + """"  
  
    ShellWait App.Path + "\link1.exe " + CmdLine  
  
    Unload Me  
End Sub  
  
Private Sub Form_Load()  
  
    If MsgBox("Linker-Controller starten?", vbYesNo) = vbNo Then  
  
        ShellWait App.Path + "\link1.exe " + Command()  
        Unload Me  
  
    End If  
  
End Sub
```

Nun kommt der kritische Teil: Erstellen Sie das Programm "Link.exe", gehen Sie danach in Ihr VB Verzeichnis und benennen Sie die Datei "Link.exe" in "Link1.exe" um, danach müssen Sie das von Ihnen erstellte Programm "Link.exe" in dieses Verzeichnis kopieren.

Wenn Sie nun im IDE das Projekt kompilieren, wird eine MessageBox gezeigt. Falls Sie "Nein" klicken, wird eine normale EXE erzeugt, ansonsten wird das Formular angezeigt.

Es muss die „ShellWait“ Funktion verwendet werden, da das IDE die erstellten Objektdaten nach dem Kompilieren sofort wieder löscht und es dann zu einem Fehler kommt.

5. Beispiel

Erstellen Sie ein neues Projekt und fügen Sie ein Modul (modFunctions) mit dem folgenden Code ein:

```
Function DLLMain(ByVal a As Long, ByVal b As Long, ByVal c As Long) As Long  
    DLLMain = 1  
End Function  
  
Sub Main()  
  
End Sub
```



```

' Dummy
End Sub

Function Subtrahieren(ByVal A As Double, ByVal B As Double) As Double
    Subtrahieren = A - B
End Function

Function Addieren(ByVal A As Double, ByVal B As Double) As Double
    Addieren = A + B
End Function

```

Die Prozedur "Main" müssen wir einfügen, damit das IDE beim Kompilieren keinen Fehler ausgibt. Beim Initialisieren der DLL wird die Funktion „DLLMain“ aufgerufen.

Nun müssen wir eine Exportdefinitionsdatei erstellen. Diese Datei gibt an, welche Funktionen exportiert werden sollen.

Erstellen Sie eine Datei namens "C:\Export.def", öffnen Sie diese mit dem Texteditor und geben Sie die folgenden Zeilen ein:

```

LIBRARY DLL
EXPORTS
    Subtrahieren
    Addieren

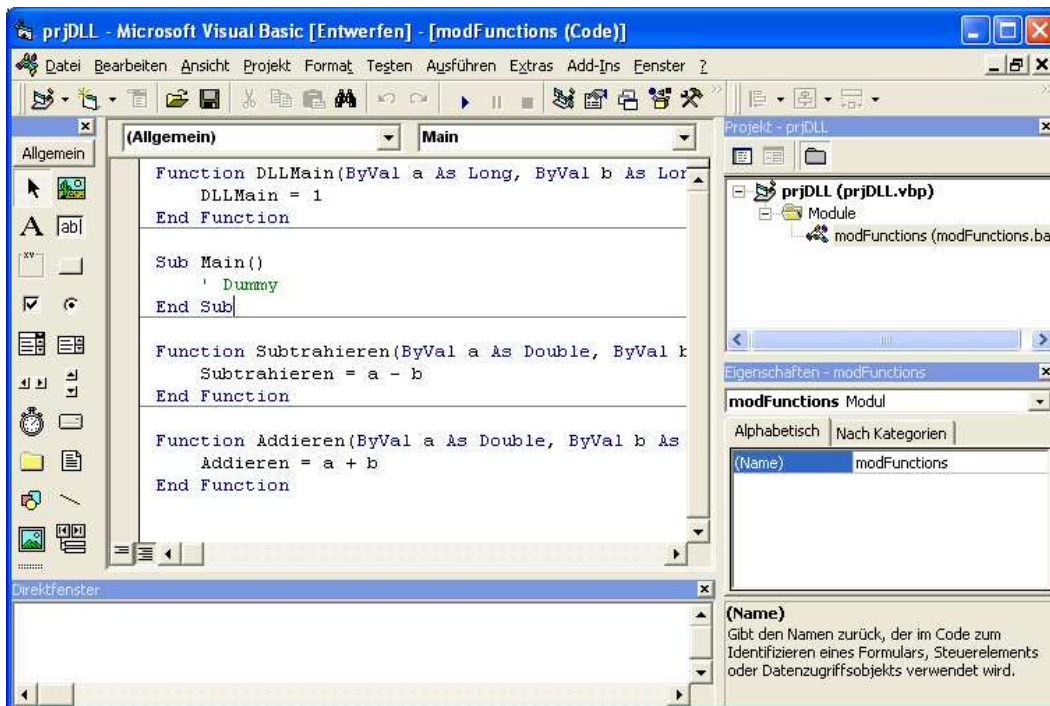
```

Hinter das Schlüsselwort "LIBRARY" müssen Sie den Namen der DLL schreiben, unter "EXPORTS" werden die exportierten Funktionen aufgelistet.

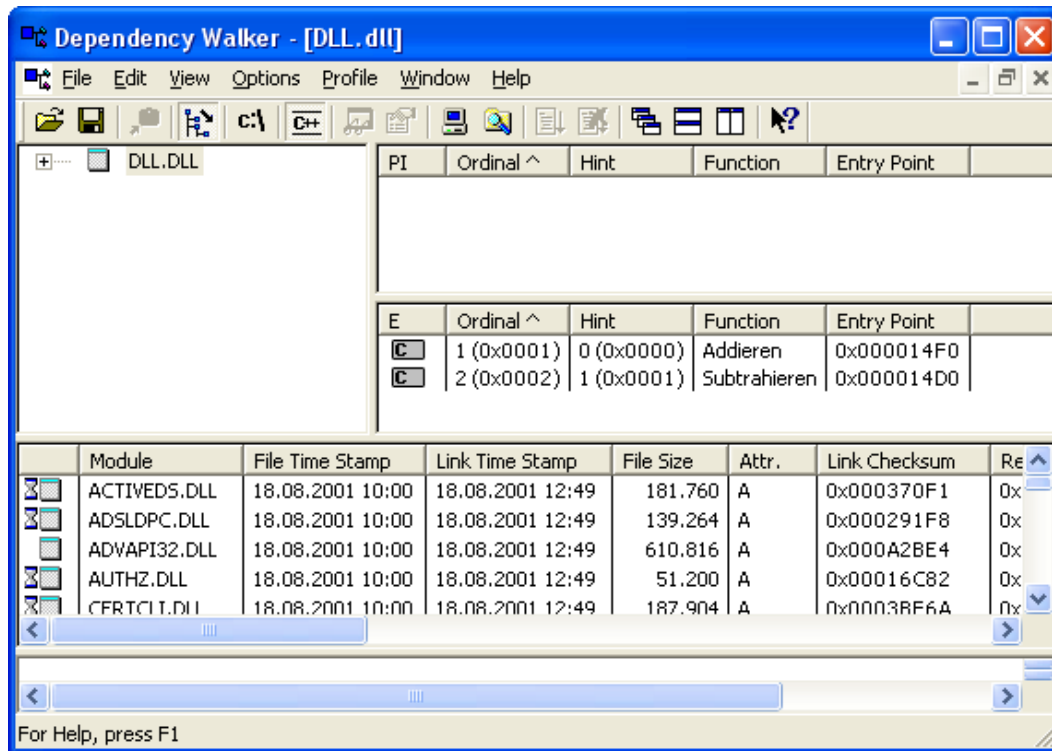
Weitere Informationen über .DEF-Dateien erhalten Sie hier:

[MSDN Modul-Definition \(.DEF\) Dateien](http://msdn.microsoft.com/library/default.asp?url=/library/en-us/vccore98/HTML/_core_module.2d.definition_files.asp) (http://msdn.microsoft.com/library/default.asp?url=/library/en-us/vccore98/HTML/_core_module.2d.definition_files.asp)

Kompilieren Sie nun das Projekt (Dateiname: "DLL.dll") und geben Sie in die Textbox des Linker-Controllers den Pfad und Dateinamen der .DEF-Datei ein (ohne Anführungszeichen) und klicken Sie auf den Button. Nun wird eine Bibliothek erstellt, die zwei Funktionen exportiert.



Dies lässt sich mit einem Programm wie z.B. dem Dependency Walker (www.dependencywalker.com) überprüfen:



6. DLL-Aufruf

Erstellen Sie eines neues Projekt fügen Sie ein Modul (modProgram) hinzu, speichern Sie dieses dann im selben Verzeichnis wie die erstellte DLL und fügen Sie den folgenden Code ein:

```
Declare Function Subtrahieren Lib "DLL.dll" (ByVal A As Double, ByVal B As Double) As Double
Declare Function Addieren Lib "DLL.dll" (ByVal A As Double, ByVal B As Double) As Double

Sub Main()

    MsgBox CStr(Addieren(3.3, 2.7))
    MsgBox CStr(Subtrahieren(3.3, 2.7))

End Sub
```

7. Bekannte Probleme

Leider treten bei DLLs einige kleine Fehler auf:

- Fehlermeldungen beim Versuch, innerhalb der exportierten Funktionen Instanzen von Klassen zu erstellen
- Probleme bei Funktionen, die als Parameter einen String erwarten oder diesen zurück geben (benutzen Sie statt dessen den Datentyp Variant)
- Bei VB6 werden die Bibliotheken teilweise nicht gefunden (fügen Sie die Declare-Statements in ein Modul ein)

8. Weitere Möglichkeiten

Natürlich können Sie den Linker-Controller noch beliebig erweitern. Durch die Angabe des Parameters `"/SUBSYSTEM:CONSOLE"` wird z.B. eine Konsolenanwendung erstellt.

Alle Möglichkeiten des Linkers ausführlich zu dokumentieren wäre eher ein Thema für ein Buch mit mehreren hundert Seiten, nicht jedoch für ein Tutorial.

Vielen Dank an Andreas Kolberg für seine Hilfe