Windows CE 3.0

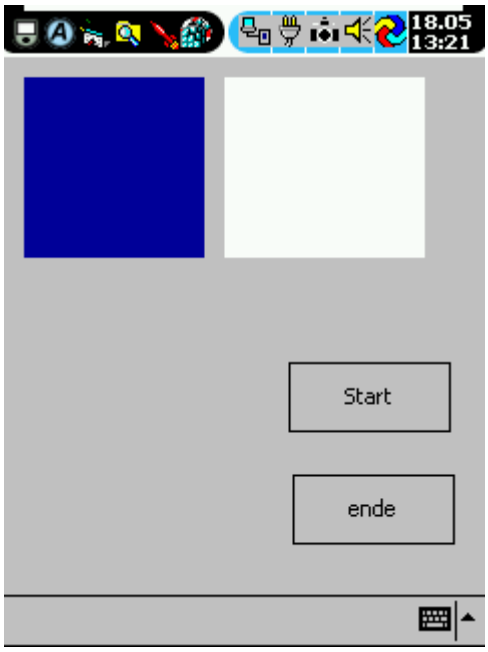
Programmiertipp: User Defined Types in CE.

Was sich Microsoft in puncto CE und UDT's ausgedacht hat, ist schwer nachzuvollziehen. Da bietet man API's an, die Strukturen aufweisen, lässt aber die entsprechenden Typendeklarationen nicht zu. Das wunderte mich am Anfang schon sehr, dass man nirgendwo eine Hilfe bekam, wie man mit diesen API's umzugehen hat. Erst nach längerer Zeit wagte ich mich dann mal wieder an diese UDT's heran. Es muss doch irgendwie möglich sein.

Ausgangspunkt war der Einsatz eines GPS – Empfängers. Das Gerät lief wunderbar über die serielle Schnittstelle. Die geografischen Werte und die GPS - Zeit konnte im Klartext dargestellt werden. Jetzt kam der Wunsch auf, diese genaue Zeitangabe zu nutzen, um die Uhr des PDA's genau abzugleichen. Und da war das Problem wieder da: UDT's müssen übergeben werden.

Mittlerweile klappt es und ich habe eine Menge dazugelernt. Ich beginne aber mit einem anderen Beispiel, das sicherlich ziemlich unsinnig ist. Hier ist es mir aber zum ersten Mal gelungen, diese UDT's umzusetzen. Der Screenshot zeigt das wahnsinnige Ergebnis an.

Zwei Kästchen – einmal blau und einmal weiß – das ist alles. Aber es hat den richtigen Weg nach vorne gezeigt, denn diese Kästchen wurden mit einer API erzeugt, die eine Strukturdefinition verlangt, einen UDT – einen User Defined Type, den es in CE eigentlich nicht gibt.



Im normalen Visual Basic benutzt man diese UDT's in der folgenden Form:

Wenn man sich den API-Aufruf FillRect ansieht, so wird hier zuerst der Devicekontext hdc, dann lprect as RECT verlangt.

Dieser RECT ist ein Typ, den man aus der API – Beschreibung herauskopieren kann.

Hier wird left, top, right und bottom deklariert. Kopieren Sie sich das kurze Programmstück und fügen es in ein leeres Visual Basic – Projekt ein.

Es muss noch ein Commandbutton eingefügt werden. Die Werte für hbrush müssen Sie einfach ausprobieren. Die 27 ergibt bei mir ein hellblaues Rechteck.

```
Private Type RECT
       Left As Long
       Top As Long
       Right As Long
       Bottom As Long
End Type

Private Declare Function FillRect Lib "user32" Alias "FillRect" (ByVal hdc As
Long, lpRect As RECT, ByVal hBrush As Long) As Long
Dim lp as RECT

Private Sub Command1_Click()
      lp.Top = 10
      lp.Left = 10
      lp.Right = 100
      lp.Bottom = 100
      r = FillRect(Me.hdc, lp, 27)
End Sub
```

Ich hoffe, dass das Beispiel problemlos geklappt hat und ein Rechteck gezeichnet werden konnte. Doch wie sieht dieses jetzt bei CE aus. Wie gesagt – Typedeklarationen sind gar nicht vorgesehen. Kontrollieren wir zunächst einmal, was dieser UDT repräsentiert. Wenn wir uns einmal ausgeben lassen: `caption = len (lp)`, so erhalten wir 16.

Es handelt sich also um einen String, der hier im Speicher abgelegt wurde. Die 16 macht auch Sinn, denn es werden 4 Longzahlen übergeben, die jeweils 4 Bytes reservieren und 4 mal 4 macht 16.

Bei Integerzahlen wären das bekanntlich zwei Bytes, die gespeichert werden. Wenn es sich um einen String handelt, so versuchen wir die API zu übertölpeln. Wir versuchen jetzt keinen Type zu übergeben, sondern direkt einen String. Dazu müssen wir aber die API – Deklarartion ändern:

aus

```
Private Declare Function FillRect Lib "user32" Alias "FillRect" (ByVal hdc As
Long, lpRect As RECT, ByVal hBrush As Long) As Long
```

wird:

```
Private Declare Function FillRect Lib "user32" Alias "FillRect" (ByVal hdc As
Long,byval lpRect As String, ByVal hBrush As Long) As Long
```

Jetzt müssen wir noch den String entwickeln. Beginnen wir einfach ganz naiv und basteln ihn zusammen. Wir hätten gerne ein Rechteck rechts=60, top=20, links=10, bottom=80.

Obwohl es nicht funktionieren wird, fangen wir an. Eine Longzahl repräsentiert 4 Bytes. Da die einzugebenden Werte alle unter 256 sind, kommen wir mit einem Wertebyte aus, die anderen Bytes füllen wir mit 0. Da ein String gemacht werden soll, müssen wir die CHR-Werte übernehmen.

```
Lpstr = string(3,0) + chr(60) + string(3,0) + chr(20) + string(3,0) + chr(20) +
string(3,0) + chr(80)

R = fillrect (me.hdc , lpstr , 27)
```

Dieses führt – wie erwartet – nicht zum Erfolg

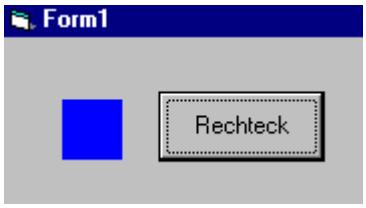
Jetzt ändern wir den String so, dass das Inhaltsbyte vorne steht und von drei Nullbytes gefolgt wird. Wir brauchen dafür nur die ersten 3 Bytes abzuschneiden und nach hinten zu versetzen.

```
Lpstr = chr(60) + string(3,0) + chr(20) + string(3,0) + chr(20) + string(3,0) +
chr(80)+
string(3,0)
R = fillrect (me.hdc , lpstr , 27)
```

Hier ist das Ergebnis zu sehen. Es ist uns tatsächlich gelungen den Benutzerdefinierten Typ UDT zu umgehen und ihn mit einem String zu ersetzen.

Dieses gibt Hoffnung, auch bei CE auf diese Weise verfahren zu können. Es ist noch einiges zu erklären, was da überhaupt gemacht wurde und natürlich müssen wir es noch einfacher machen, den UDT zu entwickeln. Achten Sie auch auf die Reihenfolge der Positionen:

Rechts, top, links, bottom.



Zunächst einmal sollte erklärt werden, warum das Verdrehen des Strings zum Erfolg führte.

Da machen wir ein paar Sätze in die Geschichte der Computer. Es könnte nämlich auch sein, dass wir den String anders hätten zusammenbauen müssen – wenn wir einen Power Mac z.B. vor uns hätten.

So leben wir mit Windows und haben Intel – basierende Maschinen.
„Windows was designed araound the little Endian architekture“ und „Little Endian first“ geben Auskunft über unser Problem. Mehr dazu am Ende des Beitrags.

Bleiben wir bei einer Integerzahl, die in 2 Bytes gespeichert wird. Diese Bytes können Werte von 0 bis 255 annehmen.

Damit können Zahlen von 255 * 256 + 255 = 65535 dargestellt werden. Wer sich noch keine Gedanken über die Darstellung solcher Zahlen gemacht hat, der kann sich hier etwas schlauer machen. Das erste Byte hat die Wertigkeit Inhalt * 256, das zweite Byte den Wert inhalt.

Die beiden Bytes 2 und 45 in der unten abgebildeten Reihenfolge (Little Endian) bedeuten : 2 45 = 2 * 256 + 45 = 557.

Die 45 im zweiten Byte ist der kleinere Anteil gegennüber der 512 im ersten Byte.

Damit ist die 45 die **Little Endian** (die kleine Endung); das erste Byte entsprechend die **Big Endian**. Der zweite Satz von oben erklärt es nun: „Little Endian first“ – das kleinere Byte wird zuerst gespeichert, dann kommt die Big Endian.

Deshalb hatten wir Erfolg, als wir die Wertebytes und Nullbytes umdrehten.

Bei Longzahlen geht es deshalb genauso. Eine Longzahl &H12345678 wird so abgespeichert: &H78 - &H56 - &H34 - &H12

Jetzt bleibt uns noch eine Funktion oder Sub zu gestalten, die uns das Umdrehen der Werte erleichtert. Dabei werde ich verschiedene Lösungsbeispiele aufzeigen.

Wir beginnen aber zuerst mit einem anderen Beispie zur Verdeutlichungl. Das Setzen der Zeit können wir zwar im regulären Visual Basic vornehmen (DATE$=“01-02-2001“ oder TIME$=“14:02:00“) – bei CE geht es nicht.

Wenn wir also in VB etwas bewirken können (ohne UDT’s zu benutzen), so haben wir die Chance, dies auch in CE zu gestalten.

Zum Setzen der Zeit gibt es zwei API – Calls Setlocaltime und Setsystemtime. Es ist eigentlich

gleich, für was wir uns entscheiden, sie sind bei uns um 2 Stunden im Sommer auseinander. Wir nehmen die Setlocaltime – Funktion.

```
Private Declare Function SetLocalTime Lib "kernel32" _
(lpSystemTime As SYSTEMTIME) As Long
```

Wir erkennen wieder ,dass hier mit einem UDT gearbeitet wird. Wir wollen das umgehen und benennen die Funktion wieder um.

```
Private Declare Function SetLocalTime Lib "kernel32" _
(ByVal lpSystemTime As String) As Long
```

Untenstehend ist der UDT aufgeführt. SYSTEMTIME hat 8 Eintäge der Art Integer, so dass der zu übergebende String 16 Bytes lang sein muss.

```
'Private Type SYSTEMTIME
'        wYear As Integer
'        wMonth As Integer
'        wDayOfWeek As Integer
'        wDay As Integer
'        wHour As Integer
'        wMinute As Integer
'        wSecond As Integer
'        wMilliseconds As Integer
'End Type
```

Dies ist ein wirklich primitives Programm. Es soll hier nur gezeigt werden, dass es funktioniert. Die Integerumwandlung und das Verdrehen der beiden Bytes wird mit der Zeile

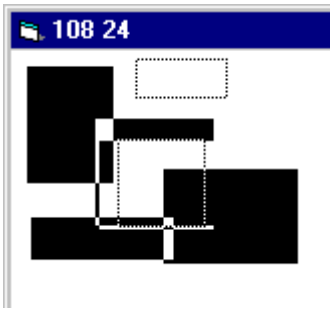
```
aus = aus + Chr(wert Mod 256) + Chr(wert \ 256)
```

vorgenommen.

```
Private Sub Command1_Click()

wert = 2001: GoSub Integer_String
wert = 5   : GoSub Integer_String
wert = 5   : GoSub Integer_String
wert = 18  : GoSub Integer_String
wert = 14  : GoSub Integer_String
wert = 29  : GoSub Integer_String
wert = 0   : GoSub Integer_String
wert = 0   : GoSub Integer_String

r = SetLocalTime(aus)
Exit Sub
Integer_String:
      aus = aus + Chr(wert Mod 256) + Chr(wert \ 256)
Return
End Sub
```

Kopieren Sie sich die wenigen Zeilen in ein leeres Visual Basic Projekt, verändern sie die Variable Wert nach dem Muster, wie es der UDT vorgibt.

Noch ein Beispiel für das Umschiffen der UDT's. Dieses wilde Bild auf der Form kann mit den unten folgenden API's erzeugt werden.



Der Ablauf ist wieder ähnlich wie oben gezeigt. Die beiden UDT's RECT in den API – Aufrufen werden wieder als String übergeben und die Longwerte werden in der Funktion lpstr nach Little Endian umgewandelt.

```
Private Declare Function DrawFocusRect Lib "User32" _
(ByVal hdc As Long, _
ByVal lpRect As String) As Long

Private Declare Function InvertRect Lib "User32" _
(ByVal hdc As Long, _
ByVal lpRect As String) As Long

Dim a, b, links, oben, rechts, unten As Integer


Private Sub Form_MouseDown(Button As Integer, _
Shift As Integer, X As Single, Y As Single)
   ScaleMode = 3
   If Button And 1 Then
      links = X
      oben = Y
      rechts = X
      unten = Y
  End If
End Sub

Private Sub Form_MouseUp(Button As Integer, _
Shift As Integer, X As Single, Y As Single)
   ScaleMode = 3
   If Not (Button And 1) Then
      a = lpstr(links, 4) + lpstr(oben, 4) + lpstr(X, 4) + lpstr(Y, 4)
        InvertRect hdc, a
   End If
End Sub

Private Sub Form_MouseMove(Button As Integer, _
Shift As Integer, X As Single, Y As Single)
   ScaleMode = 3
   Caption = X & " " & Y
   If Button And 1 Then
    DrawFocusRect hdc, a
    a = lpstr(links, 4) + lpstr(oben, 4) + lpstr(X, 4) + lpstr(Y, 4)
    DrawFocusRect hdc, a
   End If
End Sub

Function lpstr(ByVal ein As Long, art As Integer) As String
Dim i As Integer, merker As Long

merker = ein
    If ein < 0 And art = 4 Then merker = ein - &H80000000
    If ein < 0 And art = 2 Then merker = ein - &H8000
    For i = 1 To art - 1
```

```
        lpstr = lpstr & Chr(merker Mod 256)
        merker = merker \ 256
    Next i

    If ein < 0 Then
        lpstr = lpstr & Chr(merker + &H80)
           Else
        lpstr = lpstr & Chr(merker)
    End If
Print
End Function
```

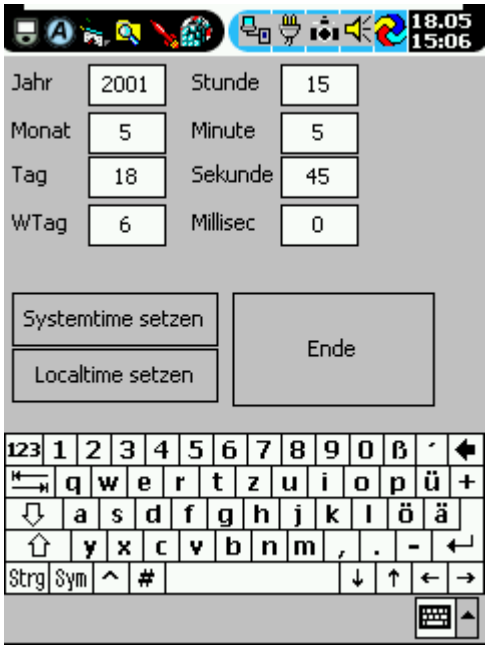
Zurück nach CE

Jetzt wird es spannend. Wird diese Methode bei CE funktionieren? Man kann sicherlich schon ahnen, dass wir Erfolg haben werden – sonst wäre der Artikel hier zu Ende. Das erste Beispiel mit dem FILLRECT ist unten abgedruckt. Es werden zwei Kästchen gezeichnet.

```
Option Explicit
Declare Function FillRect Lib "Coredll" _
(ByVal hdc As Long, ByVal lpRect As String, ByVal hBrush As Long) As Long
Dim lpstr As String

Private Sub Command1_Click()
App.End
End Sub

Private Sub Command2_Click()
Dim r
lpstr = ""
Call Long_String(10, 4)
Call Long_String(10, 4)
Call Long_String(100, 4)
Call Long_String(100, 4)
r = FillRect(Me.hdc, lpstr, 72)
lpstr = ""
Call Long_String(110, 4)
Call Long_String(10, 4)
Call Long_String(210, 4)
Call Long_String(100, 4)
r = FillRect(Me.hdc, lpstr, 5)

End Sub

Private Sub Form_Load()

End Sub

Sub Long_String(ByVal ein As Long, ByVal longorint As Integer)
Dim i As Integer
    For i = 1 To longorint - 1
        lpstr = lpstr & ChrB(ein Mod 256)
        ein = ein \ 256
    Next i
       lpstr = lpstr & ChrB(ein)
End Sub
```

Im unteren Abschnitt ist die Sub zu erkennen, die einen Longwert in 4 Bytes Little Endian umsetzt.

Das nächste Beispiel zeigt das Programm zur Einstellung von Datum und Uhrzeit.



Hier nun der Screenshot, der alles bewirkte. Nachdem ich die Systemzeit einstellen wollte, war die obige Odyssee zu durchlaufen. Mit der Umgehung des UDT's sind jetzt eine Menge Applikationen möglich, die man vorher EVB nicht realisieren konnte. Hier unten das kleine Programm, das Systemzeit und Lokalzeit einstellt. Hier ist wieder eine andere SUB gewählt, die das Umrechnen bewerkstelligt. Sie kann Integer und Longwerte verarbeiten. Die Art der Umrechnung wird beim Aufruf mitgegeben.

```
Option Explicit
Declare Function SetSystemTime Lib "Coredll" _
(ByVal lpstr As String) As Long

Declare Function SetLocalTime Lib "Coredll" _
(ByVal lpSystemTime As String) As Long

Dim lpstr As String, aus As Integer
Private Sub Command1_Click()
App.End
End Sub

Private Sub Command2_Click()

'Type SYSTEMTIME
'        wYear As Integer
'        wMonth As Integer
'        wDayOfWeek As Integer
'        wDay As Integer
'        wHour As Integer
'        wMinute As Integer
'        wSecond As Integer
'       wMilliseconds As Integer
'End Type

Call mache_udt
Call SetSystemTime(lpstr)
End Sub
Sub mache_udt()
lpstr = ""
```

```
Call NachString(Text1, 2)
Call NachString(Text2, 2)
Call NachString(Text4, 2)
Call NachString(Text3, 2)
Call NachString(Text5, 2)
Call NachString(Text6, 2)
Call NachString(Text7, 2)
Call NachString(Text8, 2)
End Sub

Private Sub Command3_Click()
Call mache_udt
Call SetLocalTime(lpstr)
End Sub

Private Sub Form_Load()
Text1 = Year(Now)
Text2 = Month(Now)
Text4 = Weekday(Now)
Text3 = Day(Now)
Text5 = Hour(Now)
Text6 = Minute(Now)
Text7 = Second(Now)
Text8 = "0"
End Sub
Sub NachString(ByVal ein As Long, ByVal art As Integer)
Dim i As Integer
    For i = 1 To art - 1
        lpstr = lpstr & ChrB(ein Mod 256)
        ein = ein \ 256
    Next i
       lpstr = lpstr & ChrB(ein)
End Sub
```

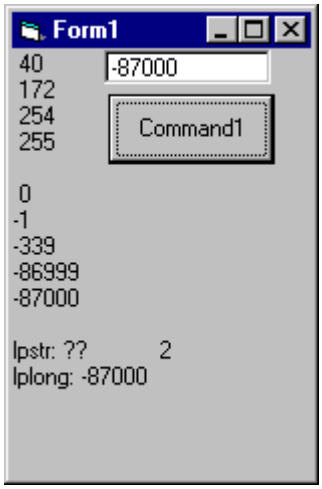
Bei der Sub fällt auf, dass statt Chr nun ChrB eingesetzt wird. Die Hilfe von VB gibt dazu folgende Information:

Anmerkung Die **ChrB**-Funktion wird mit Byte-Daten verwendet, die in einem Wert vom Typ **String** enthalten sind.. Anstelle eines Zeichens, das aus einem oder zwei Bytes bestehen könnte, gibt **ChrB** immer ein einzelnes Byte zurück. Die **ChrW**-Funktion gibt einen Wert vom Typ **String** zurück, der das Unicode-Zeichen enthält. Dies gilt nicht für Plattformen, auf denen Unicode nicht unterstützt wird. In diesem Fall entspricht das Verhalten der **Chr**-Funktion.

Neben dieser Funktion existiert auch das Prendant zu ChrB : AscB

Anmerkung Die **AscB**-Funktion wird mit Daten vom Typ **Byte** verwendet, die in einer Zeichenfolge enthalten sind. Anstelle des Zeichen-Codes für das erste Zeichen gibt **AscB** das erste Byte zurück. Die **AscW**-Funktion gibt den Unicode-Zeichen-Code zurück. Dies gilt jedoch nicht für Plattformen, auf denen Unicode nicht unterstützt wird; in diesem Fällen verhält sich **AscB** wie die **Asc**-Funktion.

Zum Umrechnen von Strings in Longzahlen und umgekehrt verwenden wir jetzt die Funktionen ChrB unde AscB. Es sind zwei Funktionen zusammengestellt, die sowohl positive wie negative Zahlen des Types Byte, Integer (zwei Byte) und Long (vier Byte) umrechenen können. Kopieren Sie sich das kleine Programm in ein leeres Visual Basic –Projekt und probieren es aus. Die print –Anweisungen können später entfernt werden. Der Screenshot zeigt die Umwandlungsschritte zu lpstr. Das Ergebnis wird wieder eingesetzt und zurückgerechnet. Aus –87000 wird wieder -87000




```
Private Sub Command1_Click()
Cls
stringwandel = lpstr(Val(Text1), 1)
longwandel = lpLong(stringwandel, 1)
Print
Print "lpstr: " & stringwandel, Len(stringwandel)
Print "lplong: " & longwandel
End Sub

Function lpstr(ByVal ein As Long, art As Integer) As String
Dim i As Integer, merker As Long

    merker = ein
    If ein < 0 And art = 4 Then merker = ein - &H80000000
    If ein < 0 And art = 2 Then merker = ein - &H8000
    For i = 1 To art - 1
        lpstr = lpstr & ChrB(merker Mod 256)
        Print merker Mod 256
        merker = merker \ 256
    Next i

    If ein < 0 Then
        lpstr = lpstr & ChrB(merker + &H80)
        Print merker + &H80
    Else
        lpstr = lpstr & ChrB(merker)
        Print merker

End If
Print
End Function

Function lpLong(ByVal ein As String, ByVal art As Integer) As Long
Dim i, merker As Integer, Neg As Boolean

    If AscB(MidB(ein, art)) And &H80 Then Neg = True
    For i = art To 1 Step -1
        merker = AscB(MidB(ein, i))

             If Neg Then merker = merker - &HFF

        lpLong = lpLong * 256 + merker
    Print lpLong
    Next i
    If Neg Then lpLong = lpLong - 1
```

```
    Print lpLong
End Function
```

Man muss ausprobieren, ob man immer mit dem ChrB gut fährt. Bei den obigen Programmen in Visual Basic hatte ich nur Erfolg mit Chr.

Anhang aus Microsofts MSDN:

SUMMARY

When designing computers, there are two different architectures for handling memory storage. They are called Big Endian and Little Endian and refer to the order in which the bytes are stored in memory. Windows NT was designed around Little Endian architecture and was not designed to be compatible with Big Endian because most programs are written with some dependency on Little Endian.

MORE INFORMATION

These two phrases are derived from "Big End In" and "Little End In." They refer to the way in which memory is stored. On an Intel computer, the little end is stored first. This means a Hex word like 0x1234 is stored in memory as (0x34 0x12). The little end, or lower end, is stored first. The same is true for a four-byte value; for example, 0x12345678 would be stored as (0x78 0x56 0x34 0x12). "Big End In" does this in the reverse fashion, so 0x1234 would be stored as (0x12 0x34) in memory. This is the method used by Motorola computers and can also be used on RISC-based computers. The RISC-based MIPS computers and the DEC Alpha computers are configurable for Big Endian or Little Endian. Windows NT works only in the Little Endian mode on both computers.

Windows NT was designed around Little Endian architecture. The Hardware Abstraction Layer (HAL) is written so that all operating system-related issues are automatically handled. Therefore, it is possible to create a HAL that could work on Big Endian architecture. The basic problem with porting the code has to do with the way the code is written for all programs. Code is often written with the assumption that Big Endian or Little Endian is being used. This may not be specific to the HAL; it could be something as simple as bit masking for graphics. To clarify this concept more, two programming examples follow.

User-defined types can contain objects, arrays, and BSTR strings, although most DLL procedures that accept user-defined types do not expect them to contain string data. If the string elements are fixed-length strings, they look like null-terminated strings to the DLL and are stored in memory like any other value. Variable-length strings are incorporated in a user-defined type as pointers to string data. Four bytes are required for each variable-length string element.

Note When passing a user-defined type that contains binary data to a DLL procedure, store the binary data in a variable of an array of the Byte data type, instead of a String variable. Strings are assumed to contain characters, and binary data may not be properly read in external procedures if passed as a String variable.