SAP Controls im Eigenbau

Erstellung von ActiveX-Controls in Visual Basic und Einbindung in SAP

Inhaltsverzeichnis

1. Einleitung 4
   1.1. Für wen ist dieses Buch? 4
   1.2. Was sind OCX/ ActiveX-Controls? 4
   1.3. Bedeutung der GUI Controls 4
   1.4. Nachteile von Controls 4
   1.5. Grenzen der SAP-Controls 4
   1.6. Möglichkeiten durch eigene Controls 4
   1.7. Nachteile von eigenen Controls 5
   1.8. Unbehandelte Themen 5
      1.8.1. SAP Dataprovider 5
      1.8.2. Drag and Drop 5
2. Grundlagen Visual Basic 5
   2.1. Entwicklungsumgebung 5
   2.2. Elemente eines Controls 6
3. Erstellung 6
   3.1. Was haben wir vor? 6
   3.2. ActiveX-Projekt anlegen 7
   3.3. Schnittstellen anpassen 9
      3.3.1. Manuell 9
      3.3.2. Mithilfe des AddIns „ActiveX-Control Interface Wizard“ 9
   3.4. Projekt hinzufügen 10
   3.5. Element zeichnen 11
   3.6. Control testen 11
   3.7. Resize-Event 12
   3.8. Nötige Methoden und Eigenschaften 13
   3.9. LISTBOX - Das Visual Basic Coding 13
      3.9.1. Event deklarieren 13
      3.9.2. Methode „Resize“ 13
      3.9.3. Methode „Clear“ 13
      3.9.4. Methoden „Mark“ und „Unmark“ 14
      3.9.5. Methode „Add_Item“ 14
      3.9.6. Methode „Delete_Item“ 14
      3.9.7. Methode „Selected_Item“ 14
      3.9.8. Methode „Get_Selected_Elemtents“ 14
      3.9.9. Methode „Get_All_Elements“ 15
      3.9.10. Eigenschaft „NumberOfItems“ 15
   3.10. BUTTONS – Das Visual Basic Coding 15
      3.10.1. Events deklarieren 15

3.10.2. Methode „Button_Click“ ... 15
3.10.3. Methode „Resize“ ... 15
3.10.4. Eigenschaft „MyCaption“ ... 16
3.10.5. Eigenschaft „MyFontsize“ ... 16
3.10.6. Eigenschaft „MyFontname“ ... 16
3.10.7. Eigenschaft „MyID“ ... 16
4. Kapselung in SAP ... 16
4.1. Anlegen der Klasse ... 16
4.1.1. CONSTRUCTOR ... 18
4.1.2. DISPATCH ... 18
4.1.3. SET_REGISTERED_EVENTS ... 19
4.2. Das SAP-Coding der Methoden: LISTBOX ... 19
4.2.1. CONSTRUCTOR ... 19
4.2.2. DISPATCH ... 20
4.2.3. SET_REGISTERED_EVENTS ... 20
4.2.4. Add_Item ... 20
4.2.5. Delete_Item ... 21
4.2.6. Get_Selected_Items ... 21
4.3. Das SAP-Coding der Methoden: BUTTON ... 21
4.3.1. CONSTRUCTOR ... 21
4.3.2. DISPATCH ... 22
4.3.3. SET_REGISTERED_EVENTS ... 23
4.3.4. Add_Item ... 24
4.3.5. Delete_Item ... 24
4.3.6. Get_Selected_Items ... 24
5. Einbindung in SAP ... 24
5.1. Testprogramm OCX-LISTBOX ... 24
5.2. Testprogramm OCX-BUTTONS ... 28
6. Fehleranalyse ... 30
6.1. Automation Queue ... 30
6.2. Trace ... 31
6.2.1. Allgemein ... 31
6.2.2. Trace einschalten ... 32
6.2.3. Trace analysieren ... 33
7. Erweiterung des Controls ... 33
7.1. Neue Eigenschaften ... 33
7.2. Funktionen ... 33
7.3. Events ... 34
8. Object Browser ... 34
9. Einbindung vorhandener Controls in SAP ... 35
10. Nutzung vorhandener Controls in Visual Basic ... 35
11. Registrierung der ActiveX-Komponente ... 35
12. MS OLE/DCOM Ressource Explorer ... 36
13. Ideen ... 37
13.1. Scannen ... 37
13.2. Technische Zeichnungen ... 37
13.3. Neues Eingabefeld ... 37
13.4. Messgeräte ... 38
13.5. Schieberegler ... 38

13.6. Schnittstellen auslesen 38
13.7. Daten per Drag'n'Drop ablegen 38
13.8. 38
14. Ausblick 38
14.1. Veraltete Technik? 38
15. Weiterführende Links 38

1. Einleitung

1.1. Für wen ist dieses Buch?

Die Implementierung von SAP-Standard-Controls setzt bereits gute Kenntnisse in der ABAP-Programmierung voraus. Für die Programmierung der Controls in Visual Basic sind ebenfalls Grundkenntnisse in dieser Sprache von Vorteil. Die Beispiele sind jedoch hinreichend einfach und werden Stück für Stück erarbeitet. Um wirklich tolle Controls schreiben zu können, sind jedoch fundierte Kenntnisse vonnöten.

Dieses Buch richtet sich an all jene, die mehr können möchten als der Großteil der SAP-Programmierer; Für alle Programmierer, die ihren Kunden eine Möglichkeit mehr bieten wollen, um eine Aufgabe umsetzen zu können. Letztendlich soll es eine Hilfestellung für die Entwickler sein, die an ungewöhnlichen Techniken und deswegen auch an ungewöhnlichen Lösungen interessiert sind.

1.2. Was sind OCX/ ActiveX-Controls?

ActiveX oder OCX ist eine Technik, um grafische Elemente und Funktionen in einer Datei zu kapseln. Die Controls müssen in Windows registriert werden. Die Controls können in vielen Umgebungen eingebunden werden: Webpages, Entwicklungsprojekten in C#, C++, Delphi etc. Die Controls bieten gekapselte Funktionen, die in anderen Umgebungen zur Verfügung stehen.

1.3. Bedeutung der GUI Controls

Mit den Controls hat die SAP die GUI-Oberfläche ins neue Jahrtausend gerettet. Die inzwischen nicht mehr wegzudenkenden Oberflächenelemente wie ALV-Grid, Text-Controls oder Controls mit Baumdarstellung sind inzwischen in viele SAP-Anwendungen integriert und machen die Bedienung der Transaktionen einfacher. Transaktionen, die diese Controls verwenden, werden auch als „Enjoy-Transaktion“ bezeichnet.

1.4. Nachteile von Controls

Die Controls machen die Eingabe von Daten oftmals einfacher und bieten eine bessere Übersichtlichkeit. Die Enjoy-Transaktionen haben jedoch den Nachteil, dass sie nicht Batchinput-fähig sind. Dies sollte immer im Hinterkopf behalten werden, wenn Controls eingebunden werden. Das soll uns aber nicht davon abhalten, unsere eigenen Controls zu entwickeln und einzubauen.

1.5. Grenzen der SAP-Controls

Controls zu programmieren ist aufwändig. SAP entwickelt die Controls jedoch immer weiter. So hat das Listbox-Control – eins der wenigen Controls, die nicht explizit eingebunden werden müssen – mehrere Entwicklungsstufen hinter sich. Es hat angefangen mit einer Dropdown-Box, die nur den Text des Feldes angezeigt hat. Inzwischen kann man den Schlüssel des Feldes einblenden und in der letzten Version auch den gewünschten Schlüssel zeichenweise eintippen.

Das ALV-Grid wird ständig um neue Funktionen erweitert: Sortierung, Eingabeverhalten, Farbgebung etc. Funktionalitäten, die hier fehlen, lassen sich nicht erweitern oder verändern.

Auch bei den von SAP ausgelieferten Controls muss man mit dem Vorlieb nehmen, was vorhanden ist. Das reicht für die meisten Fälle, aber es sind Anwendungen oder Erfordernisse denkbar, die über das hinaus gehen, was SAP vorgedacht hat.

1.6. Möglichkeiten durch eigene Controls

Auch wenn die Programmierung von Controls aufwändig ist, so lässt sich doch eine Menge von Aufgabenstellungen denken, die es notwendig machen könnten, eigene Controls implementieren zu

wollen. Einfache Funktionen lassen sich auch relativ unkompliziert erstellen. Denn auch einfache Controls können dem Anwender die Dateneingabe einfacher oder überhaupt erst möglich machen. Denken Sie an einen Gabelstaplerfahrer, der auf einem Touchscreen seine Fahrten sieht und mit den SAP-Standardknöpfen bestätigen muss… Hier können z.B. große Pushbuttons erstellt werden und auf ein Dynpro eingebunden werden. So kann der Gabelstaplerfahrer auch mit seinen Arbeitshandschuhen auf „OK“ drücken… und treffen!

Eine deutlich komplexere Aufgabe dürfte die Programmierung eines Controls sein, das z.B. eine technische Grafik – zum Beispiel eine Explosionszeichnung – darstellt, bei der der Anwender Teile auswählen kann. Die ausgewählten Teile könnten dann vom Control zurück geliefert werden, um damit eine Ersatzteillieferung zu veranlassen.

Allerdings können nicht nur solche offensichtlichen Erfordernisse den Einsatz neuer Controls nötig machen, sondern können auch eingesetzt werden, um die Akzeptanz der Anwender zu verbessern. Auf einem Kontroll-Bildschirm können z.B. Elemente eingesetzt werden, die genau so aussehen, wie die „echten Elemente“ an der Maschine. Dies kann sogar noch ein Sicherheitsaspekt sein, denn es kann einfacher erkannt werden, wenn ein Wert außerhalb eines bestimmten Bereichs liegt

1.7. Nachteile von eigenen Controls

OCX-Dateien, die die Standard-SAP-Controls enthalten, sind im SAPGUI-Verzeichnis auf jedem PC installiert. Das bedeutet, dass ein selbst geschriebenes Control ebenfalls auf die entsprechenden PCs verteilt und registriert werden muss. Das ist sicherlich keine große Hürde, sollte aber bedacht werden.

1.8. Unbehandelte Themen

Zu folgenden Themen kann ich leider keine Informationen liefern:

1.8.1. SAP Dataprovider

Die Kommunikation zwischen Control und SAP sind begrenzt. Um eine große Menge an Daten bewegen zu können, hat SAP den SAP Dataprovider entwickelt. Hierüber finden sich jedoch nur sehr wenige Informationen geschweige denn Dokumentationen. Ich kann also leider keine Infos zu dieser Funktionalität liefern.

1.8.2. Drag and Drop

Da sich meine Programmierfähigkeiten auf ABAP beschränken, finden sich in diesem Buch keine Informationen über die Behandlung von Drag’n’Drop-Funktionalitäten.

2. Grundlagen Visual Basic

Im Folgenden möchte ich einen kleinen Überblick über die Programmiersprache Visual Basic und die Entwicklungsumgebung geben. Weitere Hilfen finden Sie am Ende des Buches.

2.1. Entwicklungsumgebung

OCX-Controls werden leider nicht mehr von aktuellen Entwicklungsumgebungen unterstützt. Weder von den kostenlosen Produkten wie „Visual Basic 2008 Express“ oder SharpDevelop (#D), noch von der kostenpflichtigen IDE „Visual Basic 2008 professional“. Deswegen verwenden wir in den folgenden Beispielen die kostenlos zugängliche aber leider veraltete Entwicklungsumgebung „Visual Basic 5.0 CCE“. Wer eine professionelle Version (Visual Basic 6.0) sein eigen nennt, wird die Beispiele auch mit dieser Version entsprechend umsetzen können.

Leider bietet Microsoft den Download von VB5 nicht mehr an. Wenn Sie also nicht im Besitz von VB5 sind, müssen sie die Version 6 kaufen.

2.2. Elemente eines Controls

Ein ActiveX-Control hat im Wesentlichen die folgenden Elemente:

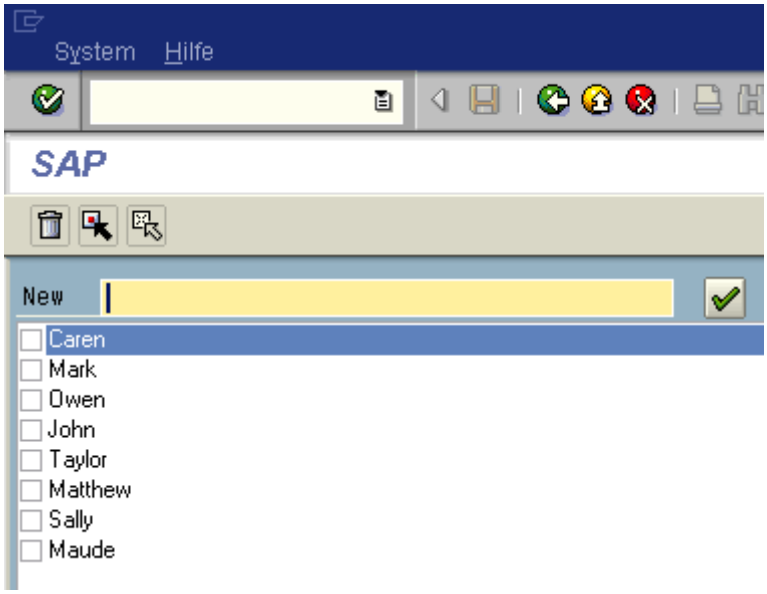
1. Ein grafisches Element, das auf der GUI präsentiert wird: Druckknopf, Eingabefeld, Liste, Regler, Viewer,…)
2. Ein Control hat in der Regel „Eigenschaften“ (Engl. Properties), die von außen beeinflussbar sind: Schriftgröße, Farbe, Sichtbarkeit, …
3. Durch „Funktionen“ werden Daten an das Control übergeben oder angefordert. Typische Funktionen (oder auch „Methoden“) sind z. B.: Aktualisieren der Anzeige, Neuberechnung, Daten hinzufügen, löschen oder ändern
4. „Hilfselemente“ unterstützen die Funktionalität eines Controls. So können beispielsweise Bilder oder Icons in einer „Bilderliste“ verwaltet werden oder ein TimerControl aktualisiert die Anzeige regelmäßig.

3. Erstellung

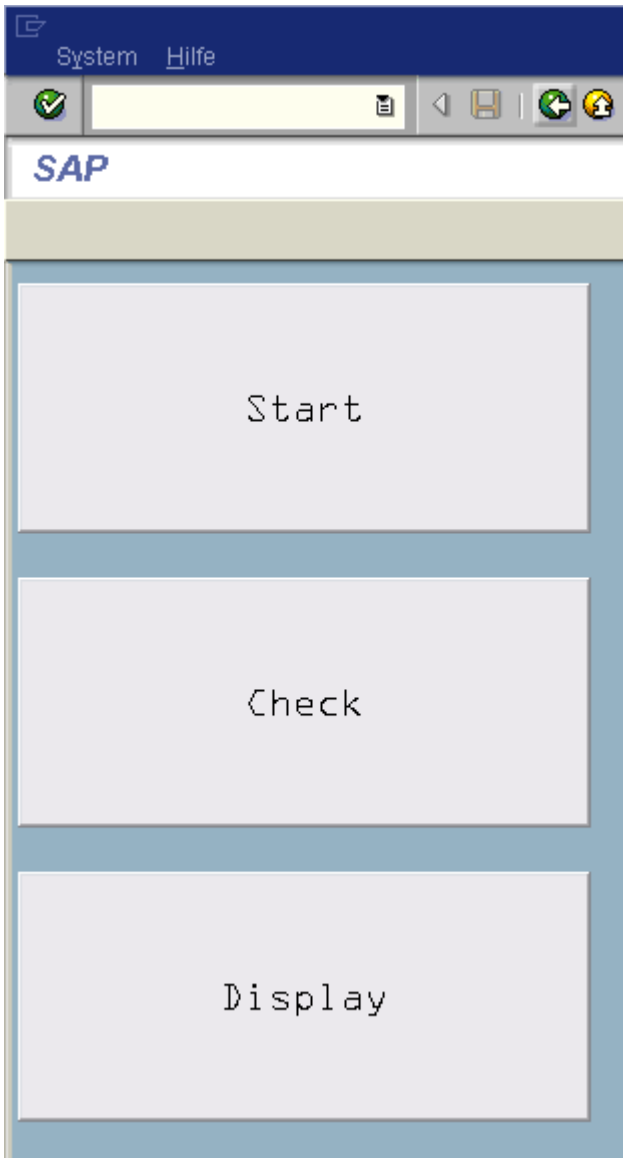
3.1. Was haben wir vor?

Wir werden zwei Beispielprojekte erstellen:

Im ersten Projekt werden wir ein Control mit dem Visual Basic Standard-Element „Listbox“ erstellen. Eine Listbox ist ein Container, in dem eine Reihe von Textelementen enthalten sind von denen eine oder mehrere ausgewählt werden können. Diese Funktionalität würde man mit SAP-Standardmitteln mit einem TableControl oder einem ALV-Grid abbilden.

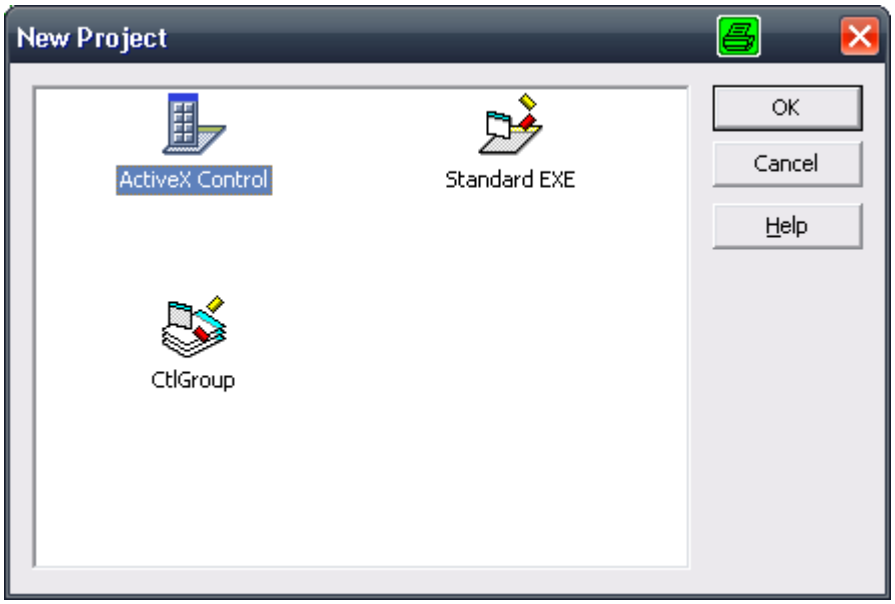


Das zweite Projekt wird einen großen „Pushbutton“ hervorbringen, der das Arbeiten an einem PC mit Touchscreen deutlich vereinfachen kann.

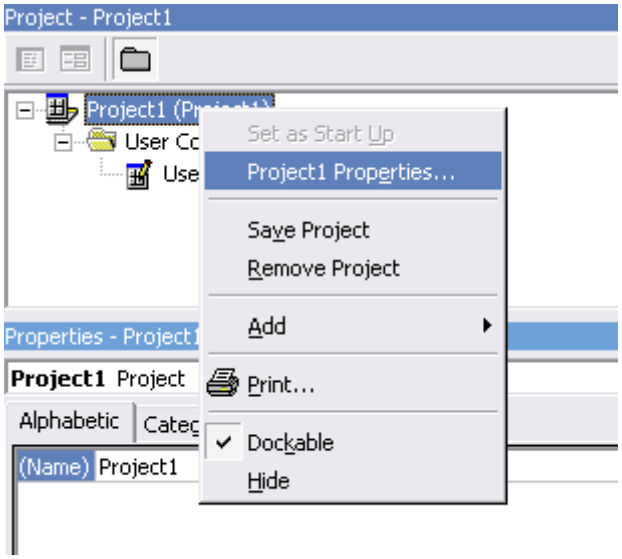


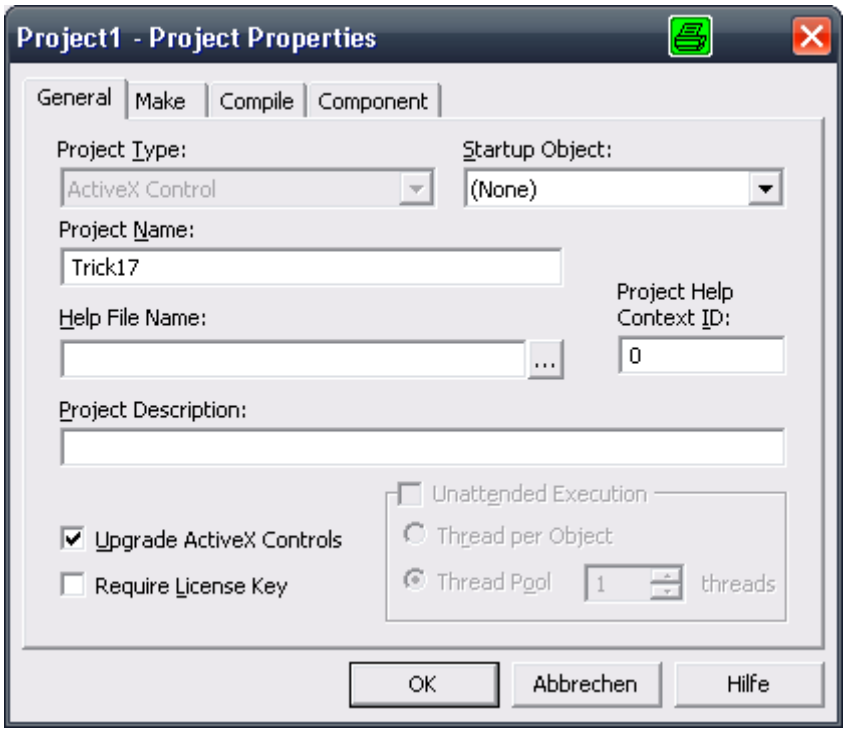
3.2. ActiveX-Projekt anlegen

Wir starten unser erstes Projekt (Menu File – New Project) und wählen das Element „ActiveX-Control“:



Danach haben wir eine freie Arbeitsfläche, die wir füllen können. Zuerst benennen wir unser Projekt aber so, dass wir es hinterher auch gut wieder finden:





3.3. Schnittstellen anpassen

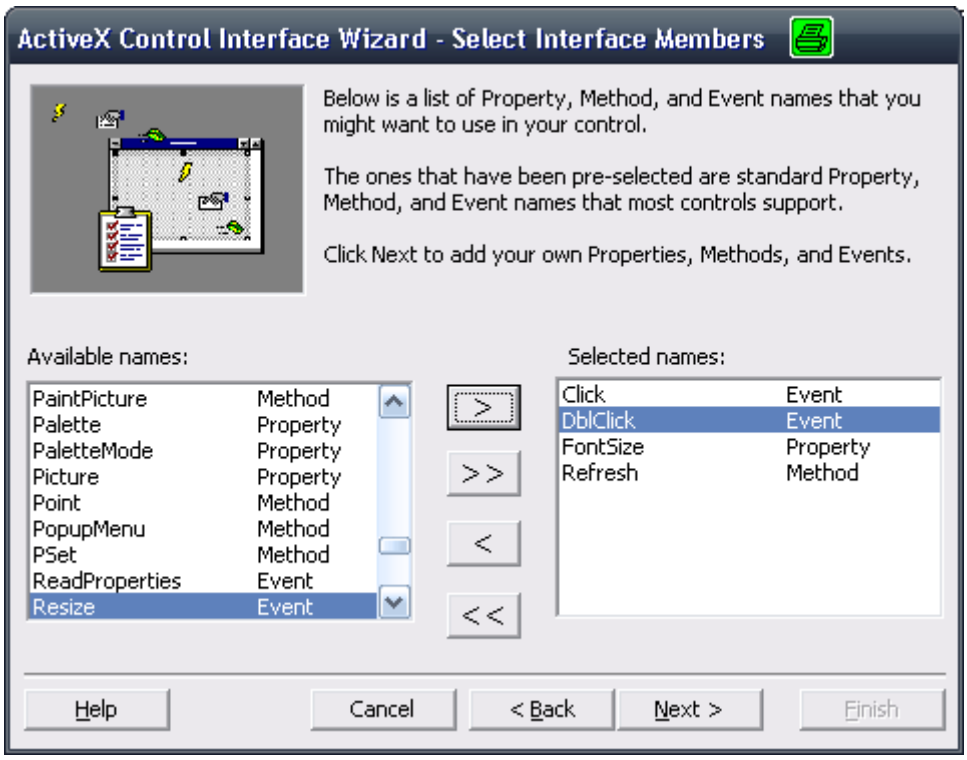
3.3.1. Manuell

Properties anlegen

Funktionen einfügen

3.3.2. Mithilfe des AddIns „ActiveX-Control Interface Wizard“

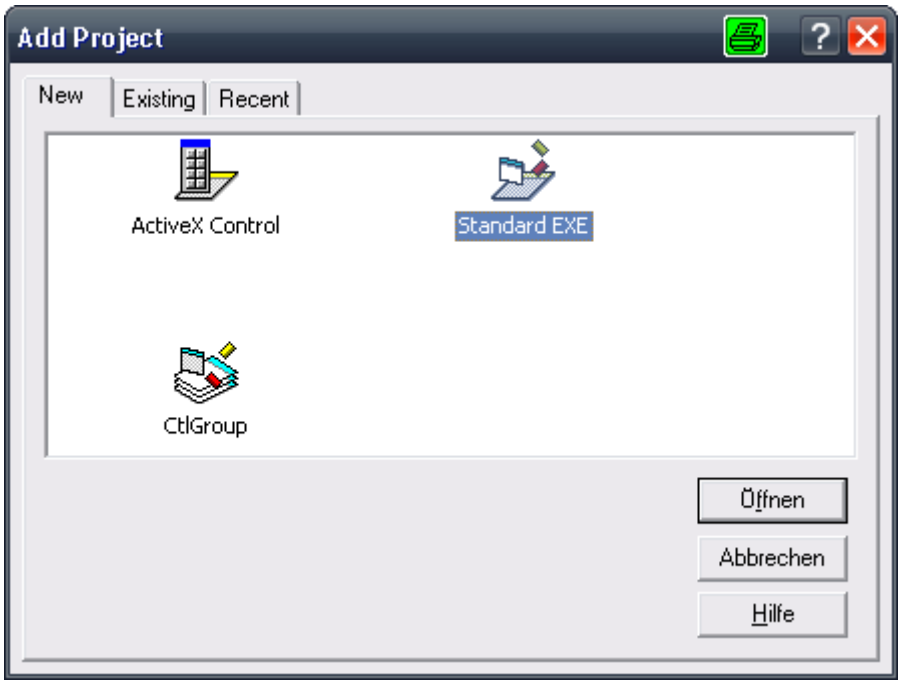
Aufruf des Wizards



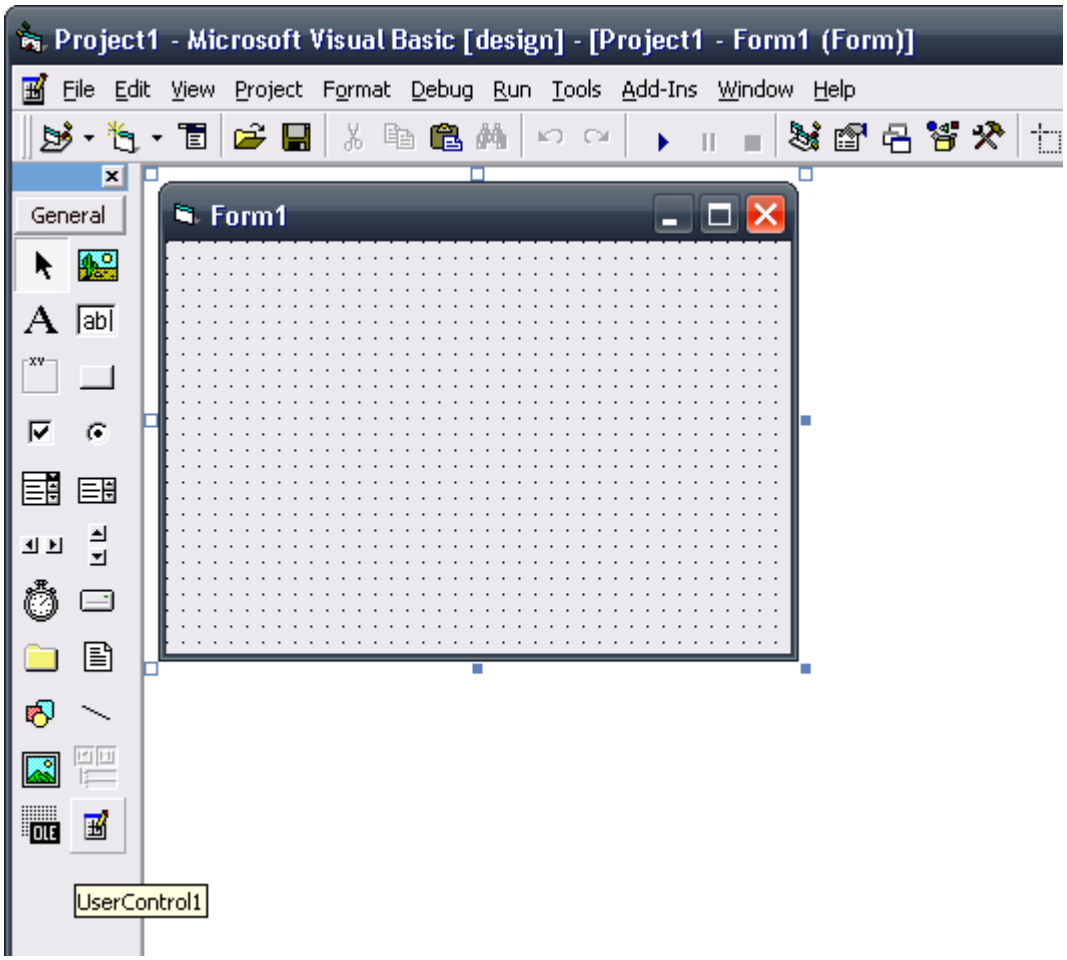
3.4. Projekt hinzufügen

Um das Control gleich testen zu können, ist es ratsam, ein neues Projekt hinzuzufügen. Dort kann das kreierte Control eingebunden und die Funktionen können gleich getestet werden.

Menü *File – Add Project*



Wir erhalten dann eine neue Arbeitsfläche auf der ein ausführbares Programm kompiliert werden kann. Dieser Schritt ist nicht zwingend notwendig, er erleichtert jedoch das Testen des Controls.



Ein Klick auf den „Start-Knopf“ (F5) kompiliert und startet das Programm. Wenn Sie das nun ausprobieren möchten, tun Sie es und Sie haben bereits ein „echtes“ Windowsprogramm zum Laufen gebracht.

3.5. Element zeichnen

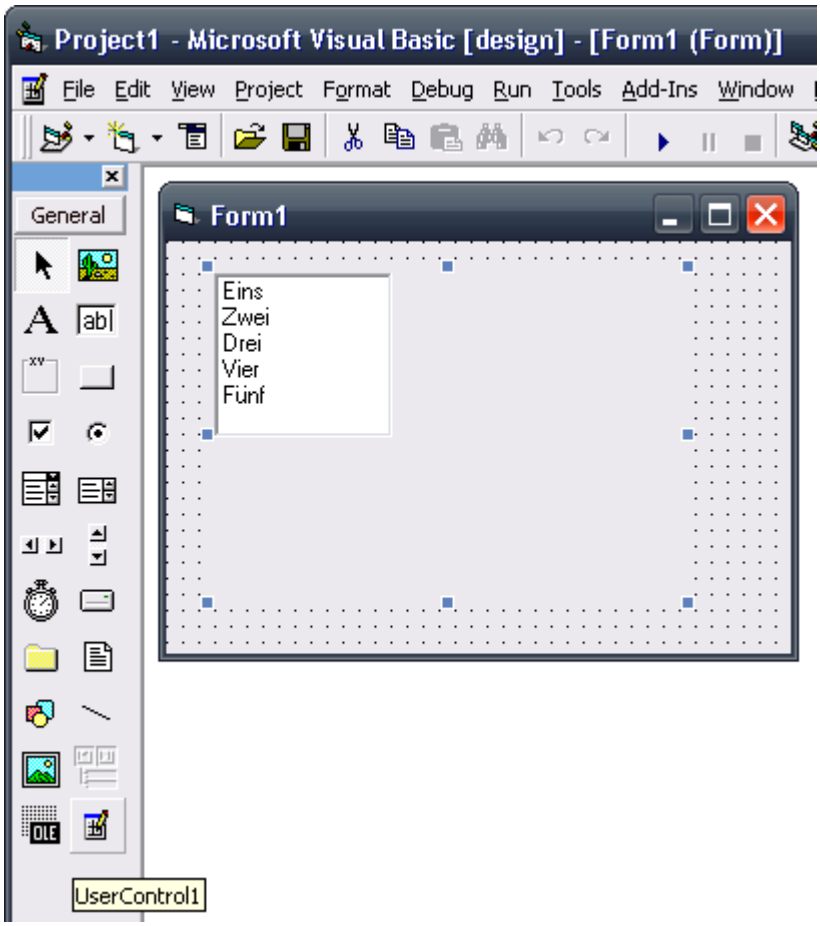
Auf der leeren Arbeitsfläche können wir nun ein grafisches Element einfügen. In der Standardpalette der verfügbaren Elemente befinden sich zwei „Components“, die es im SAP nicht gibt: Eine Listbox sowie eine Drucktaste die in beliebiger Größe erstellt werden kann.

Mit dem Zeichnen eines Elements auf der Arbeitsfläche können wir bereits ein komplett lauffähiges OCX-Control erstellen. Dieses Control hätte allerdings noch keinerlei Funktion, weswegen wir uns das Erstellen für später aufheben.

Wir binden nun unser gerade erstelltes Control auf die Arbeitsfläche unseres „EXE-Projektes“ ein.

3.6. Control testen

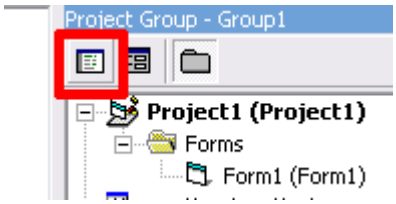
Unser gerade erstelltes Control können wir in die Arbeitsfläche des ausführbaren Programms einfügen. Um das jedoch tun zu können, müssen wir die Arbeitsfläche des Controls schliessen. Danach steht das gerade erstellte Control in der Werkzeugleiste zur Verfügung. Wir können es also anklicken und einfügen:



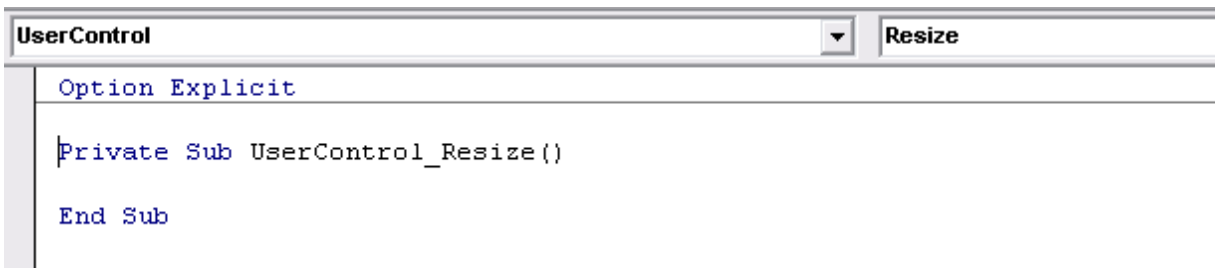
3.7. Resize-Event

Wie Sie sehen, füllt das Control die markierte Fläche nicht aus. Wir müssen also noch eine Resize-Funktionalität einbauen.

Wir wechseln dazu in den Quellcode des Controls:



Und wählen dann in den Listboxen über dem Code „UserControl“ und „Resize“. Es wird dann automatisch das folgende Coding erstellt:



Hier fügen wir dann ein:

```
If UserControl.Height <> 0 And UserControl.Width <> 0 Then
    UserControl.List1.Height = UserControl.Height
    UserControl.List1.Width = UserControl.Width

    UserControl.List1.Left = 0
    UserControl.List1.Top = 0
    UserControl.Refresh
    RaiseEvent Resize
End If
```

Das Coding ist sofort aktiv. Schließen Sie die Arbeitsflächen für das Control. Sie gelangen dann zum Programm-Projekt, auf der die Änderungen sofort sichtbar sind: das Control erweitert sich automatisch bis zur Markierung. Damit haben wir die Möglichkeit geschaffen, dass sich das Control später in der SAP-Umgebung ebenfalls an die Größe des Containers anpasst. Hierzu muss natürlich „Resizing“ im SAP Screenpainter aktiviert werden.

3.8. Nötige Methoden und Eigenschaften

So, wie das ActiveX-Control nun existiert, kann es zwar eingebunden werden, aber es ist keine Interaktion möglich. Es können also weder Werte abgefragt, noch übergeben werden. Deswegen überlegen wir uns, welche Methoden und Eigenschaften sinnvollerweise implementiert werden sollten.

Eine Listbox besitzt eine bestimmte Anzahl Elemente. Diese Elemente können „markiert“ oder „nicht markiert“ sein. Elemente müssen hinzugefügt und gelöscht werden können. Die Listbox kann die Eigenschaft „Sortiert“ haben. Das heisst, die Elemente werden beim Hinzufügen automatisch sortiert.

3.9. LISTBOX - Das Visual Basic Coding

Hier kommt nun das Visual-Basic-Coding für das ListboxControl. Soweit nötig, sind Anmerkungen zu der Methode oder zur Notwendigkeit vorhanden.

3.9.1. Event deklarieren

```
Public Event Resize()
```

3.9.2. Methode „Resize“

```
Private Sub UserControl_Resize()

If UserControl.Height <> 0 And UserControl.Width <> 0 Then
    UserControl.List1.Height = UserControl.Height
    UserControl.List1.Width = UserControl.Width

    UserControl.List1.Left = 0
    UserControl.List1.Top = 0
    UserControl.Refresh
    RaiseEvent Resize
End If

End Sub
```

3.9.3. Methode „Clear“

Hiermit löschen wir die gesamte Liste:

```
Public Sub Clear()
  UserControl.List1.Clear
End Sub
```

3.9.4. Methoden „Mark“ und „Unmark“

Diese Methoden benutzen wir, um Elemente zu markieren oder die Markierungen zu löschen. Wir benötigen dazu den Index des Elementes.

```
Public Function mark(index As Integer)
  UserControl.List1.Selected(index) = True
End Function

Public Function unmark(index As Integer)
  UserControl.List1.Selected(index) = False
End Function
```

3.9.5. Methode „Add_Item“

```
Public Function Add_Item(New_Element As String)
  List1.AddItem (New_Element)
End Function
```

3.9.6. Methode „Delete_Item“

```
Public Function Delete_Item(item As Integer)
  List1.RemoveItem (item)
End Function
```

3.9.7. Methode „Selected_Item“

Diese Methode benötigen wir, um feststellen zu können, ob ein Element markiert ist oder nicht.

```
Public Function Selected_Item(item As Integer) As Boolean
 Selected_Item = List1.Selected(item)
End Function
```

3.9.8. Methode „Get_Selected_Elemtents“

Zwischen SAP und dem Control können nur Daten einfacher Datentypen ausgetauscht werden (String, Integer,…). Um strukturierte Daten austauschen zu können, müssten wir den „SAP Dataprovider“ verwenden. Da ich hierzu keine Informationen habe, behelfen wir uns, indem wir die Liste der markierten Elemente durch Komma getrennt in einem String übergeben. Es versteht sich von selbst, dass die Elemente dann natürlich kein Komma enthalten dürfen.

```
Function Get_Selected_Elements() As String

  Dim index As Integer
  Dim list  As String
  Dim Element As String

  list = ""


  index = 0
  Do
   Element = List1.list(index)
   If Element = "" Then
     Exit Do
   Else
    If List1.Selected(index) Then
      If list = "" Then
        list = Element
      Else
        list = list + "," + Element
      End If
    End If
    index = index + 1
   End If
 Loop
```

```
 Get_Selected_Elements = list

End Function
```

3.9.9. Methode „Get_All_Elements“

Diese Methode liefert alle Elemente – markierte als auch nicht markierte – der Listbox zurück. Die Elemente werden auch hier als String und durch Komma getrennt übergeben.

```
Function Get_All_Elements() As String

  Dim index As Integer
  Dim list  As String
  Dim Element As String

  list = ""


  index = 0
  Do
   Element = List1.list(index)
   If Element = "" Then
     Exit Do
   Else
      If list = "" Then
        list = Element
      Else
        list = list + "," + Element
      End If
    index = index + 1
   End If
 Loop

 Get_All_Elements = list

End Function
```

3.9.10. Eigenschaft „NumberOfItems“

Diese Eigenschaft liefert die Anzahl der Elemente in der Liste zurück.

```
Public Property Get NumberOfItems() As Integer
  NumberOfItems = UserControl.List1.ListCount
End Property
```

3.10. BUTTONS – Das Visual Basic Coding

3.10.1. Events deklarieren

Dem Ereignis „ButtonClicked“ geben wir als Parameter die ID des Buttons mit. Wichtig hierbei: Die Parameterübergabe muss als „Wertübergabe“ (ByValue) erfolgen!

```
Event ButtonClicked(ByVal param As String)
Event Resize()
```

3.10.2. Methode „Button_Click“

```
Private Sub Button_Click()
  RaiseEvent ButtonClicked(UserControl.Button.Tag)
End Sub
```

3.10.3. Methode „Resize“

```
Private Sub UserControl_Resize()

  If UserControl.Height <> 0 And UserControl.Width <> 0 Then
    UserControl.Button.Height = UserControl.Height
    UserControl.Button.Width = UserControl.Width
```

```
    UserControl.Button.Left = 0
    UserControl.Button.Top = 0

    UserControl.Refresh
    RaiseEvent Resize
  End If

End Sub
```

3.10.4. Eigenschaft „MyCaption“

Die Beschriftung des Buttons.

```
Public Property Let MyCaption(Caption_Text As String)
  UserControl.Button.Caption = Caption_Text
End Property

Public Property Get MyCaption() As String
  MyCaption = UserControl.Button.Caption
End Property
```

3.10.5. Eigenschaft „MyFontsize“

```
Public Property Let MyFontsize(Fontsize As Integer)
  UserControl.Button.Fontsize = Fontsize
End Property

Public Property Get MyFontsize() As Integer
  MyFontsize = UserControl.Button.Fontsize
End Property
```

3.10.6. Eigenschaft „MyFontname“

```
Public Property Let MyFontname(Fontname As String)
 UserControl.Button.Font.Name = Fontname
End Property

Public Property Get MyFontname() As String
 MyFontname = UserControl.Button.Font.Name
End Property
```

3.10.7. Eigenschaft „MyID“

```
Public Property Get MyID() As String
  MyID = UserControl.Button.Tag
End Property

Public Property Let MyID(id As String)
  UserControl.Button.Tag = id
End Property
```

4. Kapselung in SAP

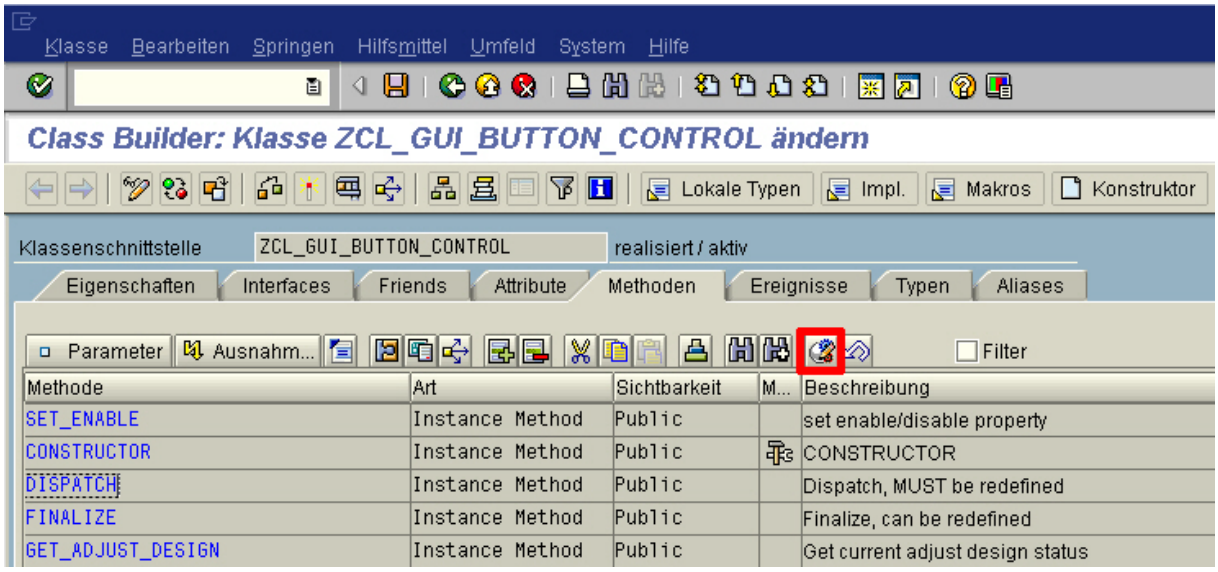
Hier wird es nun ernst. Wir binden unsere Controls in eine Klasse ein. Dies ermöglicht es uns, das Control später in ein Dynpro einzubinden.

4.1. Anlegen der Klasse

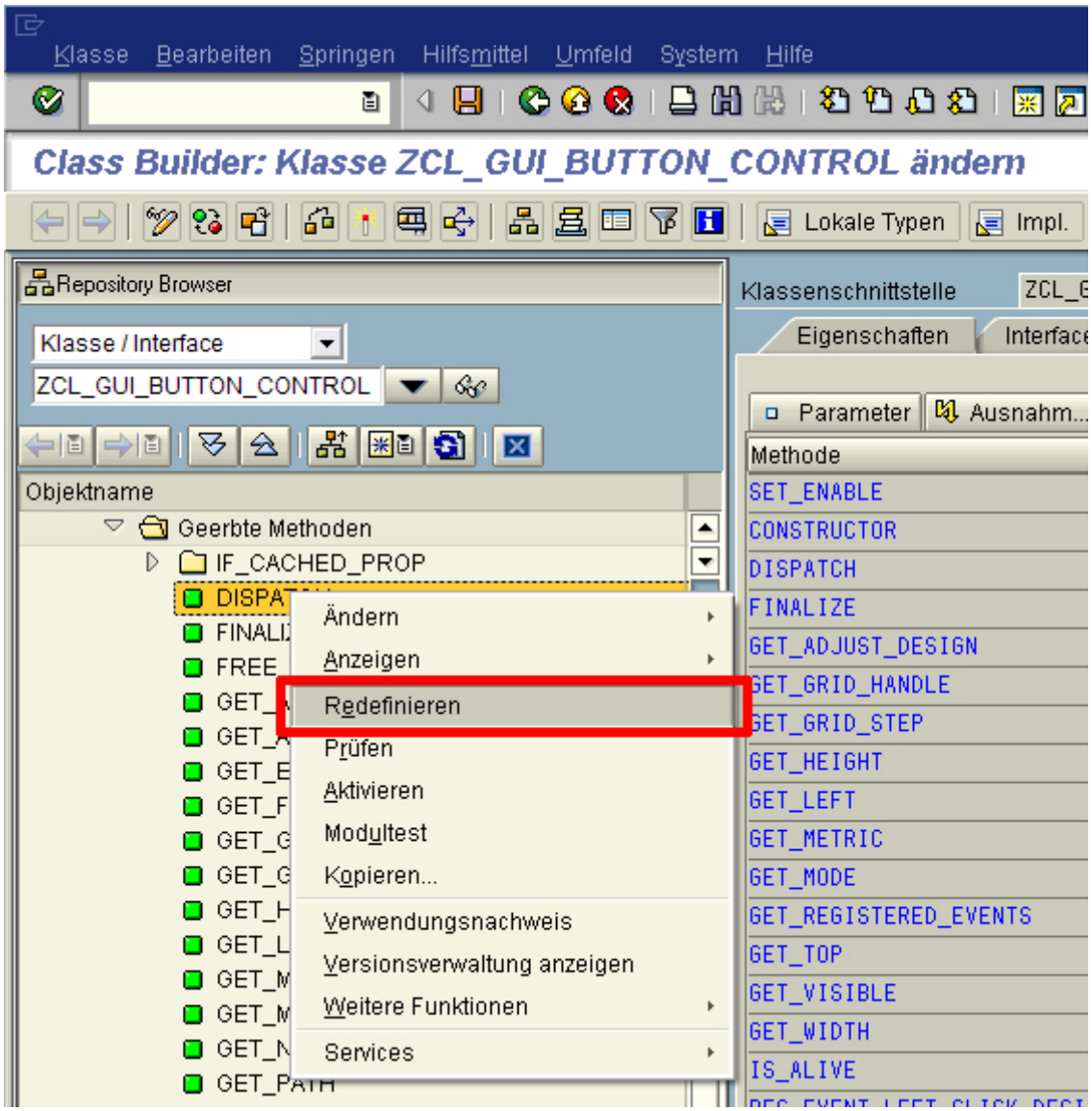
Um die Funktionen des in Visual Basic erstellten OCX-Controls nutzen zu können, müssen wir die verwendeten Eigenschaften und Funktionen in einer Klasse kapseln.

In der SE24 legen wir dazu eine Klasse an: ZCL_GUI_LISTBOX_CONTROL für die Listbox und ZCL_GUI_BUTTON_CONTROL für das ButtonControl und lassen Sie von der Klasse CL_GUI_CONTROL erben. Damit steht uns das Grundgerüst für die Kapselung zur Verfügung.

Um das OCX-Control verwenden zu können, müssen wir die folgenden drei Methoden redefinieren. Klicken Sie dazu in der SE24 die Methode an und klicken Sie dann auf den Button „redefinieren“:



Wenn Sie die SE80 verwenden, klicken Sie im Navigator mit der rechten Maustaste auf die Methode und wählen Sie „Redefinieren“:

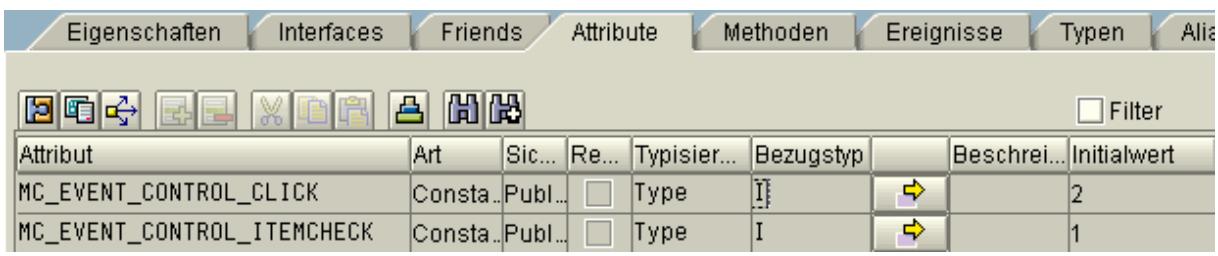


4.1.1. CONSTRUCTOR

Hier wird das Control bekannt gemacht und sozusagen mit SAP verknüpft. Die „PROG_ID“ muss identisch mit dem Projektnamen und Namen des Controls, getrennt durch einen Punkt, sein. In unserem Fall heißt das Projekt „Trick17“, Das Control heißt „Listbox1“. Die „PROG_ID“ muss also „Trick17.Listbox1“ heißen. Um die „PROG_ID“ korrekt zu ermitteln, sehen Sie bitte in Kapitel 12 „MS OLE/DCOM Ressource Explorer“ nach.

4.1.2. DISPATCH

Die Methode Dispatch erhält die Ereignisse des Control die dann dieses Ereignis in der Klasse auslösen. Das Resizing-Event müssen wir nicht abfangen, da es nicht im SAP-Programm ausgewertet wird. Die Event-ID sollte in der Klasse in den Attributen definiert werden:



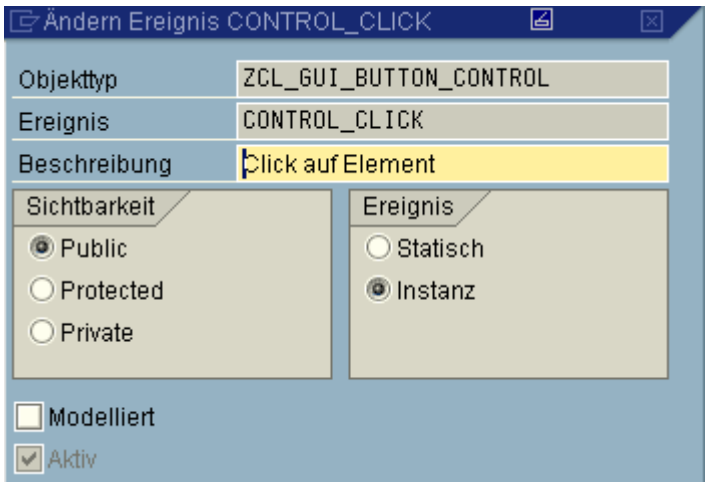
4.1.3. SET_REGISTERED_EVENTS

In dieser Methode werden die im Control möglichen Events registriert. Die Methode wird aus dem behandelnden Programm aufgerufen um zu definieren, auf welche Events das Programm reagieren möchte.

Die im Beispiel verwendete Konstante „mc_event_control_click“ muss als Attribut in der Klasse definiert werden. Im Visual Basic Coding erhalten die Events ihre Nummern in der Reihenfolge der Deklaration:

```
Public Event Event_one() '-> Event-ID „1“
Public Event Event_two() '-> Event-ID „2“
```

Das Event selbst muss natürlich ebenfalls in der Klasse definiert werden:



4.2. Das SAP-Coding der Methoden: LISTBOX

4.2.1. CONSTRUCTOR

```
METHOD constructor.

  DATA prog_id(80).

  IF parent IS INITIAL.
    RAISE error_cntl_create.
  ENDIF.

  CLASS cl_gui_cfw DEFINITION LOAD.

* assign prog_id to get the frontend specific control
  IF NOT activex IS INITIAL.
    prog_id = 'Trick17.Listbox1'.
  ELSEIF NOT javabean IS INITIAL.
    RAISE gui_type_not_supported.
  ENDIF.

  IF prog_id IS INITIAL.
    RAISE gui_type_not_supported.
  ENDIF.

* Set the window styles of the control when style parameter was not
* set with constructor call.
* For more information on the styles see WIN32 SDK
  DATA  style TYPE i.
* otherwise the control would be invisible and the mistake would be
```

```
* hard to find
  style = cl_gui_control=>ws_visible
        + cl_gui_control=>ws_child
        + cl_gui_control=>ws_clipsiblings.

* Create the control
  CALL METHOD super->constructor
    EXPORTING
      clsid      = prog_id
      shellstyle = style
      parent     = parent
      lifetime   = lifetime
      name       = name
    EXCEPTIONS
      OTHERS     = 1.

  IF sy-subrc <> 0.
    RAISE error_cntl_create.
  ENDIF.

* register instance at framework
  CALL METHOD cl_gui_cfw=>subscribe
    EXPORTING
      shellid = h_control-shellid
      ref     = me
    EXCEPTIONS
      OTHERS  = 1.

  IF sy-subrc <> 0.
    RAISE error_cntl_create.
  ENDIF.

* create and initialize dataprovider =>  m_dp_handle
  CALL FUNCTION 'DP_CREATE'
    CHANGING
      h_dp             = m_dp_handle
    EXCEPTIONS
      dp_create_error  = 1
      dp_install_error = 2
      dp_error         = 3
      OTHERS           = 4.
  IF sy-subrc <> 0.
    RAISE error_dp_create.
  ENDIF.

ENDMETHOD.
```

4.2.2. DISPATCH

Für die Listbox benötigen wir die Methode DISPATCH nicht, da das Control keine Ereignisse hat.

4.2.3. SET_REGISTERED_EVENTS

Für die Listbox benötigen wir die Methode SET_REGISTERED_EVENTS nicht, da das Control keine Ereignisse hat.

4.2.4. Add_Item

Um Elemente in die Listbox einfügen zu können, müssen wir es schaffen, die im Control implementierte Funktion „Add_Item“ anzusprechen. Wir übergeben dazu einen Namen für das Element. Dieser Name wird dann als Element in die Listbox eingefügt.

```
method add_item.

  call method call_method
    exporting
```

```
      method  = 'Add_Item'
      p_count = 1
      p1      = i_element.

  call method cl_gui_cfw=>flush.

endmethod.
```

4.2.5. Delete_Item

Einträge zu löschen wäre ebenfalls sinnvoll… Wir benutzen dazu den Index des Elements.

```
method del_item.


  call method call_method
    exporting
      method  = 'Delete_Item'
      p_count = 1
      p1      = i_index.

  call method cl_gui_cfw=>flush.

endmethod.
```

4.2.6. Get_Selected_Items

Für die Rückgabetabelle ET_ITEMS legen wir uns einen Tabellentyp an, der als Zeilentyp INT4 hat.

Parameter	Art	Typisierung	Bezugstyp	Beschreibung
ET_ITEMS	Exporting	Type	ZINT4_T	Tabelle mit Integer-Werten

```
METHOD GET_SELECTED_ITEMS.

  data lv_index type i.

  clear et_items.

  gv_number_of_items = number_of_items( ).

  DO gv_number_of_items TIMES.
    lv_index = sy-index - 1.
    gv_item_selected = get_selected( i_index = lv_index   ).
    IF gv_item_selected <> 0. " state_dead.
      APPEND lv_index TO et_items.
    ENDIF.
  ENDDO.

  sort et_items DESCENDING.

ENDMETHOD.
```

4.3. Das SAP-Coding der Methoden: BUTTON

4.3.1. CONSTRUCTOR

```
METHOD constructor.

  DATA prog_id(80).

  IF parent IS INITIAL.
    RAISE error_cntl_create.
  ENDIF.

  CLASS cl_gui_cfw DEFINITION LOAD.
```

```
* assign prog_id to get the frontend specific control
  IF NOT activex IS INITIAL.
    prog_id = 'Trick17.Listbox1'.
  ELSEIF NOT javabean IS INITIAL.
    RAISE gui_type_not_supported.
  ENDIF.

  IF prog_id IS INITIAL.
    RAISE gui_type_not_supported.
  ENDIF.

* Set the window styles of the control when style parameter was not
* set with constructor call.
* For more information on the styles see WIN32 SDK
  DATA  style TYPE i.
* otherwise the control would be invisible and the mistake would be
* hard to find
  style = cl_gui_control=>ws_visible
        + cl_gui_control=>ws_child
        + cl_gui_control=>ws_clipsiblings.

* Create the control
  CALL METHOD super->constructor
    EXPORTING
      clsid      = prog_id
      shellstyle = style
      parent     = parent
      lifetime   = lifetime
      name       = name
    EXCEPTIONS
      OTHERS     = 1.

  IF sy-subrc <> 0.
    RAISE error_cntl_create.
  ENDIF.

* register instance at framework
  CALL METHOD cl_gui_cfw=>subscribe
    EXPORTING
      shellid = h_control-shellid
      ref     = me
    EXCEPTIONS
      OTHERS  = 1.

  IF sy-subrc <> 0.
    RAISE error_cntl_create.
  ENDIF.

* create and initialize dataprovider =>  m_dp_handle
  CALL FUNCTION 'DP_CREATE'
    CHANGING
      h_dp             = m_dp_handle
    EXCEPTIONS
      dp_create_error  = 1
      dp_install_error = 2
      dp_error         = 3
      OTHERS           = 4.
  IF sy-subrc <> 0.
    RAISE error_dp_create.
  ENDIF.

ENDMETHOD.
```

4.3.2. DISPATCH

```
METHOD dispatch.
```

```
*** be sure that control really exists
  IF me->h_control IS INITIAL.
    RAISE cntl_error.
  ENDIF.

  CASE eventid.
    WHEN me->mc_event_control_click.
      RAISE EVENT control_click.

    WHEN OTHERS.
***  we don't care
  ENDCASE.

ENDMETHOD.
```

4.3.3. SET_REGISTERED_EVENTS

In dieser Methode werden die im Control möglichen Events registriert. Die Methode wird aus dem behandelnden Programm aufgerufen um zu definieren, auf welche Events das Programm reagieren möchte.

Die in diesem Beispiel verwendete Konstante „mc_event_control_click“ muss als Attribut in der Klasse definiert werden. Im Visual Basic Coding erhalten die Events ihre Nummern in der Reihenfolge der Deklaration:

Public Event Event_one() '-> Event-ID „1“
Public Event Event_two() '-> Event-ID „2“

Hier das Coding für das Bekanntmachen der Events:

```
method set_registered_events.

  data: simple_event type cntl_simple_event, "// event
        ex_event     type cntl_event,        "// eventid, is_shellevent
        events_ex    type cntl_events.       "// table

*   map simple_event into ex_event, append to events_ex
  loop at events into simple_event.
    case simple_event-eventid.
      when mc_event_control_click.
        ex_event-eventid = mc_event_control_click.
        ex_event-is_shellevent = ' '.
*         check for system / application event
        if simple_event-appl_event is initial.
          ex_event-is_systemevent = 'X'.
        endif.
        append ex_event to events_ex.
      when others.
        raise illegal_event_combination.
    endcase.

  endloop.

  call method me->set_registered_events_ex
    exporting
      eventtab                 = events_ex
    exceptions
      cntl_error               = 1
      cntl_system_error        = 2
      illegal_event_combination = 3
      others                   = 4.
```

```
  case sy-subrc.
    when 0.
    when 1. raise cntl_error.
    when 2. raise cntl_system_error.
    when 3. raise illegal_event_combination.
    when others. raise cntl_error.
  endcase.

  registered_simple_events[] = events.

endmethod.
```

4.3.4. Add_Item

Um Elemente in die Listbox einfügen zu können, müssen wir es schaffen, die im Control implementierte Funktion „Add_Item“ anzusprechen. Wir übergeben dazu einen Namen für das Element. Dieser Name wird dann als Element in die Listbox eingefügt.

```
method add_item.

  call method call_method
    exporting
      method  = 'Add_Item'
      p_count = 1
      p1      = i_element.

  call method cl_gui_cfw=>flush.

endmethod.
```

4.3.5. Delete_Item

Einträge zu löschen wäre ebenfalls sinnvoll… Wir benutzen dazu den Index des Elements.

```
method del_item.


  call method call_method
    exporting
      method  = 'Delete_Item'
      p_count = 1
      p1      = i_index.

  call method cl_gui_cfw=>flush.

endmethod.
```

4.3.6. Get_Selected_Items

5. Einbindung in SAP

Bisher haben wir nur die Grundlagen geschaffen, um das Control darzustellen und anzusprechen. Wir schreiben nun ein kleines Programm, in dem das Control in ein Dynpro (CustomContainer) eingebunden wird und testen die Methoden und Ereignisse.

5.1. Testprogramm OCX-LISTBOX

```
*---------------------------------------------------------------------*
* Report  ZZLISTBOXOCX
*---------------------------------------------------------------------*
REPORT  zzlistboxocx.
```

```
DATA gr_container TYPE REF TO cl_gui_custom_container.
DATA gr_listbox   TYPE REF TO zcl_listboxocx.
DATA gv_entries   TYPE string.
DATA gt_items     TYPE crmt_bsp_int_tab.
FIELD-SYMBOLS <item> LIKE LINE OF gt_items.
DATA gt_elements  TYPE swhttext80.
DATA gv_new_item  TYPE string.

CLASS lcl_listbox DEFINITION DEFERRED.

DATA g_application TYPE REF TO lcl_listbox.

*---------------------------------------------------------------------*
*       CLASS lcl_listbox DEFINITION
*---------------------------------------------------------------------*
CLASS lcl_listbox DEFINITION.

  PUBLIC SECTION.

    METHODS:
      handle_control_click      FOR EVENT control_click      OF zcl_listboxocx,
      handle_control_item_check FOR EVENT control_item_check OF zcl_listboxocx
                                IMPORTING item.


ENDCLASS.                    "lcl_listbox DEFINITION

*---------------------------------------------------------------------*
*       CLASS lcl_listbox IMPLEMENTATION
*---------------------------------------------------------------------*
CLASS lcl_listbox IMPLEMENTATION.

  METHOD handle_control_click.
    MESSAGE i000(oo) WITH 'CLICK JUCHU'.
  ENDMETHOD.                    "handle_control_click

  METHOD handle_control_item_check.
    MESSAGE s000(oo) WITH item.
  ENDMETHOD.                    "handle_control_item_check
ENDCLASS.                    "lcl_listbox IMPLEMENTATION

START-OF-SELECTION.

  CALL SCREEN 100.

END-OF-SELECTION.


*&---------------------------------------------------------------------*
*&      Module  STATUS_0100  OUTPUT
*&---------------------------------------------------------------------*
MODULE status_0100 OUTPUT.

  SET PF-STATUS '100'.
  PERFORM init_controls.

ENDMODULE.                 " STATUS_0100  OUTPUT
*&---------------------------------------------------------------------*
*&      Module  USER_COMMAND_0100  INPUT
*&---------------------------------------------------------------------*
MODULE user_command_0100 INPUT.

  CASE sy-ucomm.
    WHEN 'BACK'.
      SET SCREEN 0.
      LEAVE SCREEN.
```

```
    WHEN 'TEST'.
      CALL METHOD gr_listbox->test.

    WHEN 'DELETE'.

      CALL METHOD gr_listbox->get_selected_items
        IMPORTING
          et_items = gt_items.
      LOOP AT gt_items ASSIGNING <item>.

        CALL METHOD gr_listbox->del_item
          EXPORTING
            i_index = <item>.

      ENDLOOP.

    WHEN 'SHOW'.

      CALL METHOD gr_listbox->get_all_elements
        IMPORTING
          entries = gt_elements.
*      BREAK-POINT.

*      CALL METHOD gr_listbox->get_list
*        IMPORTING
*          entries = gv_entries.
*      MESSAGE s000(oo) WITH gv_entries.
    WHEN 'MARK'.
      CALL METHOD gr_listbox->mark_items.

    WHEN 'UNMARK'.
      CALL METHOD gr_listbox->unmark_items.

    WHEN 'SEL'.
      CALL METHOD gr_listbox->get_selected_items
        IMPORTING
          et_items = gt_items.
*      BREAK-POINT.

    WHEN 'ADD' OR space.
      CHECK gv_new_item IS NOT INITIAL.

*      call method gr_listbox->add_list
*       exporting
*          i_element = 'ENNO'.
*      CALL METHOD gr_listbox->get_list
*        IMPORTING
*          entries = gv_entries.

      CALL METHOD gr_listbox->add_item
        EXPORTING
          i_element = gv_new_item.
      CLEAR gv_new_item.

*      cl_gui_cfw=>flush( ).
      .
*      call METHOD gr_listbox->get_list
*        importing entries = gv_entries.
*      MESSAGE s000(oo) WITH gv_entries.
  ENDCASE.

ENDMODULE.                 " USER_COMMAND_0100  INPUT

*&---------------------------------------------------------------------*
*&      Form  init_controls
*&---------------------------------------------------------------------*
```

```
*       text
*----------------------------------------------------------------------*
FORM init_controls.


  IF gr_container IS INITIAL.

    CREATE OBJECT gr_container
      EXPORTING
        container_name = 'CONTAINER'
      EXCEPTIONS
        OTHERS         = 6.

    CREATE OBJECT gr_listbox
      EXPORTING
        parent                 = gr_container
        name                   = 'Listbox'
      EXCEPTIONS
        gui_type_not_supported = 1
        error_cntl_create      = 2
        error_dp_create        = 3
        OTHERS                 = 4.
    IF sy-subrc = 0.
      CREATE OBJECT g_application.
      PERFORM set_handler.
      cl_gui_cfw=>flush( ).
*      gv_entries = gr_listbox->get_list( ).

    ENDIF.


  ENDIF.

ENDFORM.                    "init_controls

*&---------------------------------------------------------------------*
*&      Form  set_handler
*&---------------------------------------------------------------------*
FORM set_handler.

***Set Event handler
  SET HANDLER g_application->handle_control_click FOR gr_listbox.

  DATA: my_simple_event  TYPE cntl_simple_event.
  DATA: my_simple_events TYPE cntl_simple_events.

****Register Events with CFW
*  my_simple_event-eventid = gr_listbox->mc_event_control_click.
** Specify that it's an application event
*  my_simple_event-appl_event = 'X'.
*  APPEND my_simple_event TO my_simple_events.

****Register Events with CFW
  my_simple_event-eventid = gr_listbox->mc_event_control_itemcheck.
* Specify that it's an application event
  my_simple_event-appl_event = 'X'.
  APPEND my_simple_event TO my_simple_events.

* Redefine SET_REGISTERED_EVENTS
  CALL METHOD gr_listbox->set_registered_events
    EXPORTING
      events                    = my_simple_events
    EXCEPTIONS
      illegal_event_combination = 1
      OTHERS                    = 2.

ENDFORM.                    "set_handler
```

5.2. Testprogramm OCX-BUTTONS

```
*---------------------------------------------------------------------*
* Report  ZZBUTTONOCX
*---------------------------------------------------------------------*
REPORT  zzbuttonocx.

DATA gr_container  TYPE REF TO cl_gui_custom_container.
CLASS lcl_button   DEFINITION DEFERRED.

DATA gr_button01   TYPE REF TO lcl_button.
DATA gr_button02   TYPE REF TO lcl_button.
DATA gr_button03   TYPE REF TO lcl_button.

*---------------------------------------------------------------------*
*       CLASS lcl_button DEFINITION
*---------------------------------------------------------------------*
*
*---------------------------------------------------------------------*
CLASS lcl_button DEFINITION.
  PUBLIC SECTION.
    DATA lr_container TYPE REF TO cl_gui_custom_container.
    DATA lr_button    TYPE REF TO zcl_button_ocx.

    METHODS:
      constructor IMPORTING iv_container_name TYPE string
                            iv_button_caption TYPE string
                            iv_button_id      TYPE string,

    handle_control_click      for event control_click      of zcl_button_ocx
                              importing param sender.

ENDCLASS.                    "lcl_button DEFINITION


*---------------------------------------------------------------------*
*       CLASS lcl_button IMPLEMENTATION
*---------------------------------------------------------------------*
CLASS lcl_button IMPLEMENTATION.

  METHOD constructor.

    DATA my_simple_event   TYPE cntl_simple_event.
    DATA my_simple_events  TYPE cntl_simple_events.
    DATA my_container_name TYPE c LENGTH 50.

    IF lr_container IS INITIAL.

      my_container_name = iv_container_name.
      CREATE OBJECT lr_container
        EXPORTING
          container_name = my_container_name
        EXCEPTIONS
          OTHERS         = 6.

      CREATE OBJECT lr_button
        EXPORTING
          parent                 = lr_container
          name                   = iv_container_name
          caption                = iv_button_caption
          id                     = iv_button_id
        EXCEPTIONS
          gui_type_not_supported = 1
          error_cntl_create      = 2
          error_dp_create        = 3
          OTHERS                 = 4.
      IF sy-subrc = 0.
```

```
***Set Event handler
        SET HANDLER handle_control_click FOR lr_button.


****Register Events with CFW
        my_simple_event-eventid = lr_button->mc_event_control_click.
* Specify that it's an application event
        my_simple_event-appl_event = 'X'.
        APPEND my_simple_event TO my_simple_events.

* Redefine SET_REGISTERED_EVENTS
        CALL METHOD lr_button->set_registered_events
          EXPORTING
            events                    = my_simple_events
          EXCEPTIONS
            illegal_event_combination = 1
            OTHERS                    = 2.

        cl_gui_cfw=>flush( ).

      ENDIF.


    ENDIF.

  ENDMETHOD.                    "constructor

  METHOD handle_control_click.
    MESSAGE s000(oo) WITH param.
  ENDMETHOD.                    "handle_control_click


ENDCLASS.                    "lcl_button IMPLEMENTATION


START-OF-SELECTION.

  CALL SCREEN 100.

END-OF-SELECTION.


*&---------------------------------------------------------------------*
*&      Module  STATUS_0100  OUTPUT
*&---------------------------------------------------------------------*
MODULE status_0100 OUTPUT.

  SET PF-STATUS '100'.
  PERFORM init_controls.

ENDMODULE.                 " STATUS_0100  OUTPUT
*&---------------------------------------------------------------------*
*&      Module  USER_COMMAND_0100  INPUT
*&---------------------------------------------------------------------*
MODULE user_command_0100 INPUT.

  CASE sy-ucomm.
    WHEN 'BACK'.
      SET SCREEN 0.
      LEAVE SCREEN.
  ENDCASE.

ENDMODULE.                 " USER_COMMAND_0100  INPUT

*&---------------------------------------------------------------------*
*&      Form  init_controls
```

```
*&---------------------------------------------------------------------*
FORM init_controls.

  IF gr_button01 IS INITIAL.
    CREATE OBJECT gr_button01
      EXPORTING
        iv_container_name = 'BUTTON01'
        iv_button_caption = 'Start'
        iv_button_id      = 'UCOMM_START'.
  ENDIF.

  IF gr_button02 IS INITIAL.
    CREATE OBJECT gr_button02
      EXPORTING
        iv_container_name = 'BUTTON02'
        iv_button_caption = 'Check'
        iv_button_id      = 'UCOMM_CHECK'.
  ENDIF.

  IF gr_button03 IS INITIAL.
    CREATE OBJECT gr_button03
      EXPORTING
        iv_container_name = 'BUTTON03'
        iv_button_caption = 'Display'
        iv_button_id      = 'UCOMM_DISPLAY'.
  ENDIF.


ENDFORM.                    "init_controls
```

6. Fehleranalyse

6.1. Automation Queue

Die Automation Queue steuert den Datenaustausch zwischen Frontend und SAP-System. Die Daten und Anweisungen, die an ein Control geschickt oder vom Control empfangen werden, werden hier gepuffert. Deswegen muss nach vielen Anweisungen ein „Flush“ (cl_gui_cfw=>flush) erfolgen, der die Automation Queue anweist, die Daten an das Frontend zu senden.

Wenn das Control einen Fehler liefert, dann erhält man in der Regel den wenig aussagekräftigen Shortdump „MESSAGE_TYPE_X“ bei der Methode „FLUSH“:

Ausschnitt Quelltext

Zeile	Quelltext
1	
2	METHOD FLUSH .
3	* ...
4	data: rc like sy-subrc, msgli like sy-msgli.
5	
6	CALL FUNCTION 'AC_FLUSH_CALL'
7	EXPORTING
8	SYSTEM_FLUSH = ' '
9	IMPORTING
10	MESSAGE_NR = rc
11	MESSAGE_TEXT = msgli.
12	
13	sy-subrc = rc.
14	sy-msgli = msgli.
15	
16	CASE SY-SUBRC.
17	WHEN 0.
18	WHEN 1.
19	* system_error
20	MESSAGE ID 'CNDP' TYPE 'X' NUMBER 007 RAISING CNTL_SYSTEM_ERROR.
21	WHEN 2.
22	* method_call_error
23	MESSAGE ID 'CNDP' TYPE 'X' NUMBER 006 RAISING CNTL_ERROR.
24	WHEN 3.
25	* property_set_error
>>>>>	MESSAGE ID 'CNDP' TYPE 'X' NUMBER 006 RAISING CNTL_ERROR.
27	WHEN 4.
28	* property_get_error
29	MESSAGE ID 'CNDP' TYPE 'X' NUMBER 006 RAISING CNTL_ERROR.
30	WHEN OTHERS.
31	RAISE CNTL_ERROR.
32	ENDCASE.
33	ENDMETHOD.

Genauere Informationen bekommt man, wenn man den Trace einschaltet.

6.2. Trace

6.2.1. Allgemein

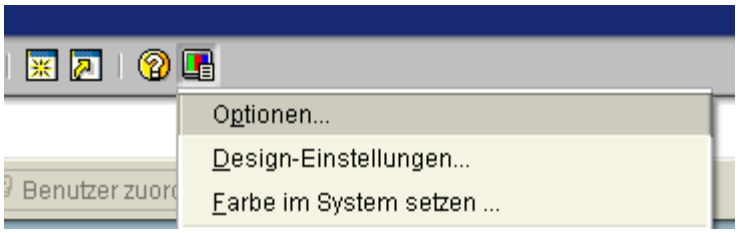
Der Trace der Automation-Queue sollte immer dann eingeschaltet werden, wenn ein Control einen „sehr allgemeinen Fehler“ zurückliefert. In dem Trace werden die Datenströme aufgezeichnet und können später ausgewertet werden.

Wenn Sie also einen Shortdump aufgrund eines Fehlers in einem Control erhalten, dann gehen Sie folgendermaßen vor:

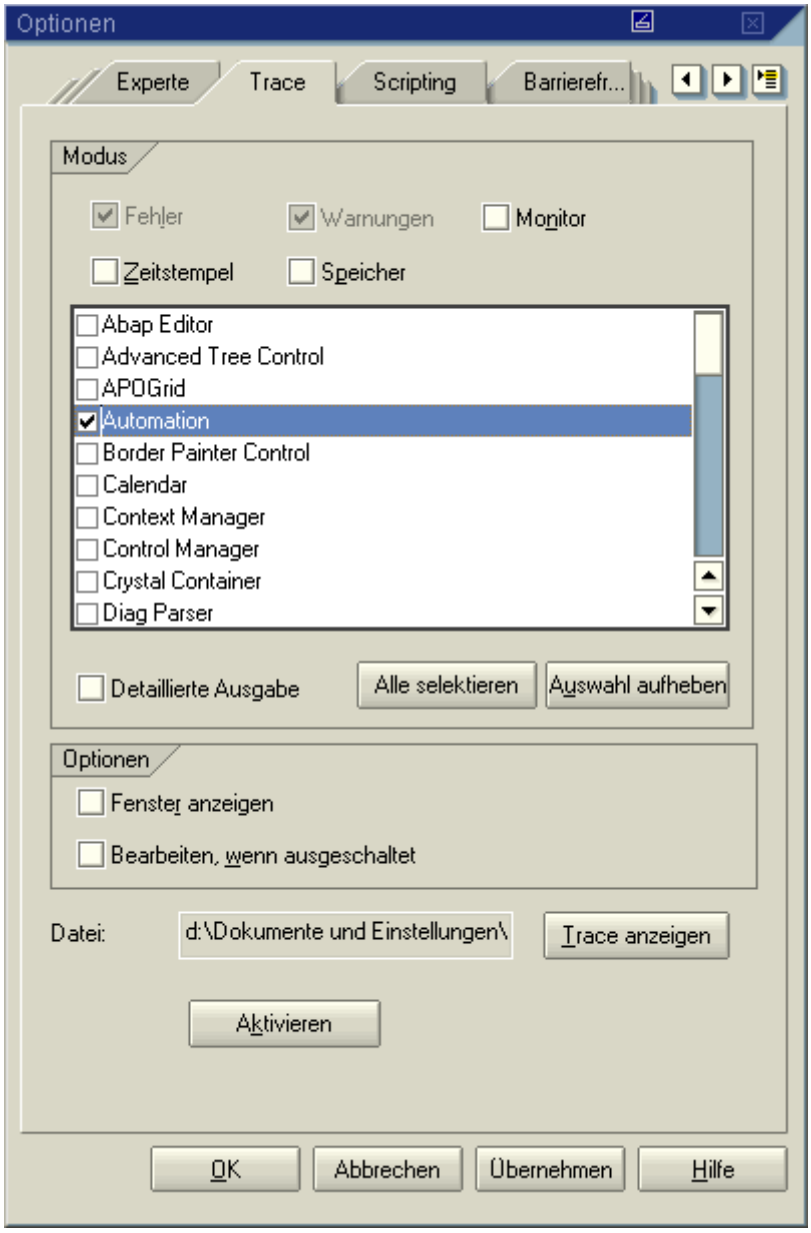
1. Aufruf der Transaktion oder des Programms und zwar bis zu der Stelle, **bevor** der Fehler auftritt
2. Trace aktivieren
3. Im Programm fortfahren bis der Dump ausgelöst wird.
4. Traceprotokoll analysieren

6.2.2. Trace einschalten

Der Trace wird im Systemmenü eingeschaltet:



Reiter „Trace“; Checkbox „Automation“ anklicken und die Änderung mit „Aktivieren“ übernehmen:



Starten Sie das Programm, welches den Fehler ausgelöst hat. Es werden alle Aktionen der Automation Queue protokolliert.

Hinweis: Wenn Sie „Ärger“ mit einem bestimmten SAP-Control haben, dann können Sie hierfür explizit den Trace aktivieren. Es werden dann nur Datenströme zu diesem Control aufgezeichnet.

6.2.3. Trace analysieren

Nachdem der Kurzdump erscheint, wechseln Sie wieder in das Optionen-Menü und klicken beim Reiter „Trace“ auf „Deaktivieren“. Klicken Sie dann auf „Trace anzeigen“. Es öffnet sich der Texteditor mit dem Protokoll welches in etwa so aussehen könnte:

```
*****************************************************************************
***************************  SAP AG, Trace File  ****************************
*****************************************************************************

Date:                      Thu Nov 26 11:35:20 2009

Trace mode (0x103):        Warning Error (Components)
*****************************************************************************
Trace components: Advanced Tree Control
*****************************************************************************
************** data to server ***************
************** data from server ***************
************** data to server ***************
************** data from server ***************
<1140=Warning:    Active-X shellId 0 tried to fire event 3 while token is at server => skipped event
<1140=Warning:    Active-X shellId 0 tried to fire event 10 while token is at server => skipped event
<1140=Warning:    Active-X shellId 0 tried to fire event 13 while token is at server => skipped event
<1140=Warning:    Active-X shellId 0 tried to fire event 24 while token is at server => skipped event
<1140=Warning:    Active-X shellId 0 tried to fire event 24 while token is at server => skipped event
<1140=Advanced Tree Control:    Creating Tree Control
<1140=Warning:    Active-X shellId 116 tried to fire event 16 while token is at server => skipped event
<1140=Warning:    Active-X shellId 116 tried to fire event 31 while token is at server => skipped event
<1140=Warning:    Active-X shellId 116 tried to fire event 1 while token is at server => skipped event
<1140=Warning:    Active-X shellId 116 tried to fire event 27 while token is at server => skipped event
<1140=Warning:    Active-X shellId 116 tried to fire event 50 while token is at server => skipped event
<1140=Advanced Tree Control:    Start Processing 10 nodes for insert
<1140=Advanced Tree Control:    Start Processing 37 items for insert
<1140=Advanced Tree Control:    ######### REPORTING ERROR #############
<1140=Advanced Tree Control:    Item already exists      LastError node key  : KONDM_02         LastError item
name : 3
<1140=Advanced Tree Control:    ########################################
<1140=(Error):    CALL METHOD "AddItemTable"[DispID=26] OF [#10/0x0E4076B8/202/SAP.TableTreeControl.1]
                    #0: OBJECT [#13/0x0C973DB4/202/{799104C5-04C5-11D0-AC56-080009D23306}]
IDispatch::Invoke raised exceptionAusnahmefehler aufgetreten
<1140=(Error):

<1140=(Error):    ****************************ERROR OCCURED IN MODULE: [SAP Table Tree Control
]*************************************************************************************************************
<1140=(Error):    PROGRAM_ID                |MODULE_NAME                |METHOD_NAME      |ERROR DESCRIPTION
|VERSION           |GUI VERSION        |MODULE_PATH                                  |
<1140=(Error):
*************************************************************************************************************************
*****************************************************************
<1140=(Error):    SAP.TableTreeControl.1     |SAP Table Tree Control      |AddItemTable     |Item already exists
|7100.4.14.304     |7100.4.14.1046     |c:\PROGRA~1\sap\frontend\sapgui\wdttree.ocx     |
<1140=(Error):
*************************************************************************************************************************
*****************************************************************
<1140=(Error):
<1140=(Error):    Exception fire by :wdttree
<1140=(Error):    Exception info:Item already exists
<1140=(Error):    Exception code:0
<1140=:
************** data to server ***************
************** data from server ***************
<1141=:
************** data to server ***************
************** data from server ***************
************** data from server ***************
```

In obigem Beispiel wurde einem TreeControl (CL_GUI_LIST_TREE) ein „Item“ doppelt übergeben. Dies lässt sich im Fehlerprotokoll recht schnell nachvollziehen.

7. Erweiterung des Controls

7.1. Neue Eigenschaften

Durch Coding in Visual Basic neue Eigenschaften mittels PROPERTY LET und PROPERTY GET definieren und deren Einbindung in SAP

7.2. Funktionen

Einbindung einer Funktion im OCX-Control und Einbindung in SAP

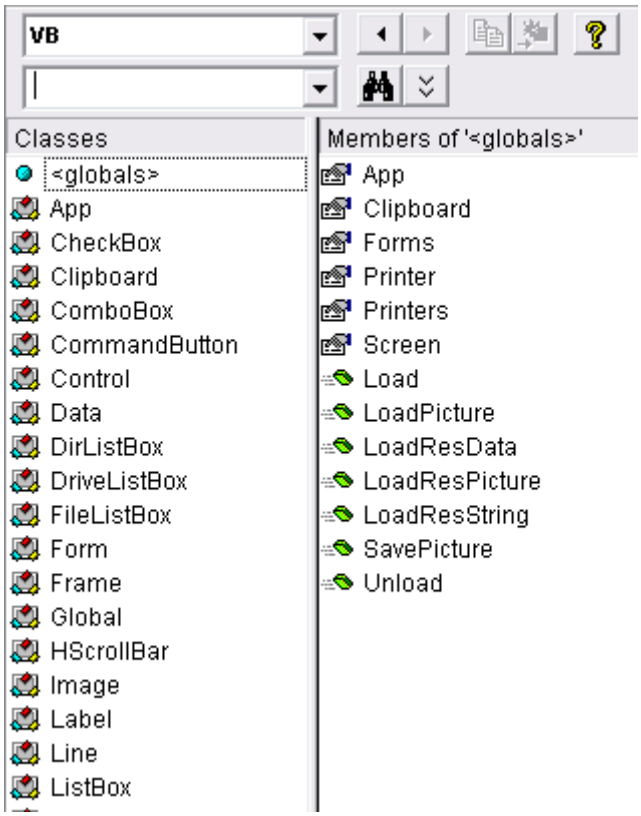
7.3. Events

Auslösen eines Events mit und ohne Parameter und Abfrage in SAP.

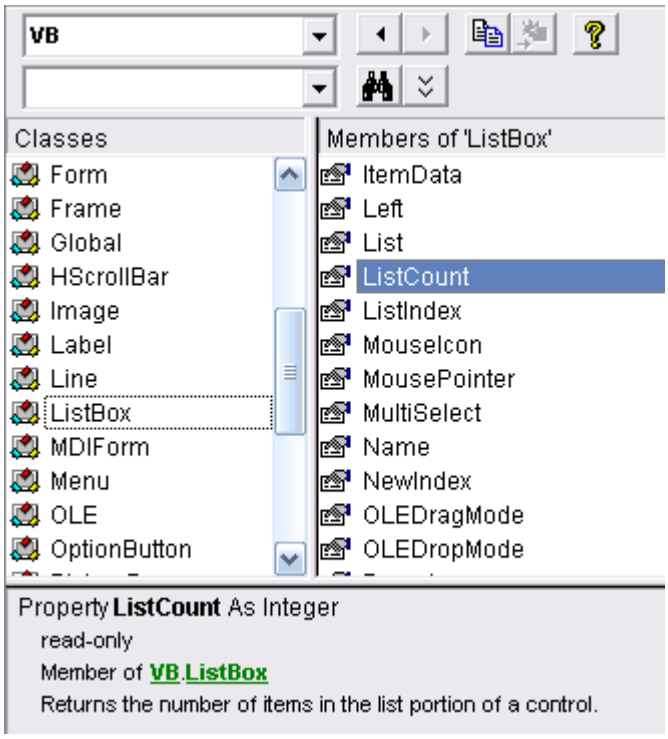
8. Object Browser

Welche Eigenschaften und Methoden in den Elementen vorhanden sind, kann der „Object Browser“ verraten. Er wird in der Entwicklungsumgebung mit der Taste F2 aufgerufen (Menu: View – Object Browser).

Wählen Sie dann die Bibliothek („Library“) aus, in der das gewünschte Element vorhanden ist, z.B. „VB“:



Klicken Sie dann links die gewünschte Klasse an und rechts die entsprechende Methode oder Eigenschaft. Im unteren Bereich sehen Sie dann eine kurze Erklärung:



9. Einbindung vorhandener Controls in SAP

Einbindung vorhandener Controls in SAP; Ermittlung der Funktionen und Eigenschaften

10. Nutzung vorhandener Controls in Visual Basic

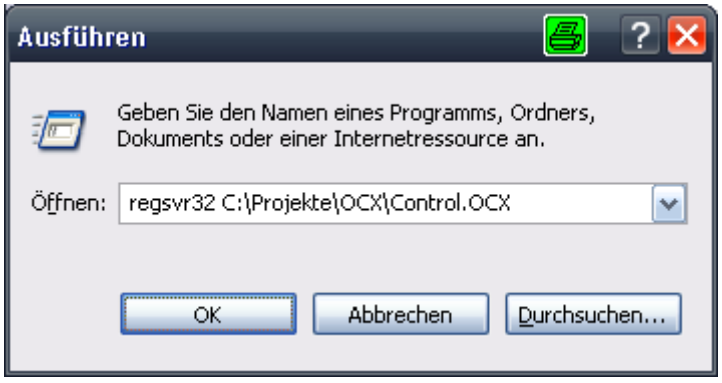
Auf dem PC registrierte Controls ins VB Projekt einbinden.

11. Registrierung der ActiveX-Komponente

Beim Erstellen der OCX-Datei in Visual Basic wird das OCX-Control automatisch registriert. Das bedeutet, dass das Control von der Windows-Bibliothek verwaltet wird.

Falls die OCX-Datei auf anderen Rechnern zum Einsatz kommen soll, muss die Registrierung manuell erfolgen.

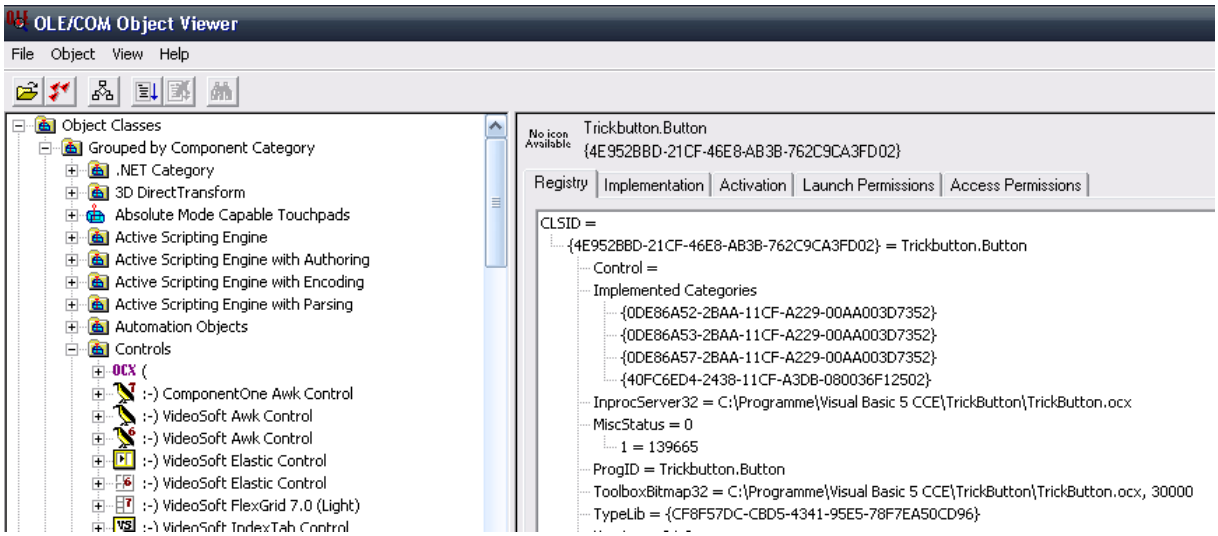
Dazu öffnen Sie bitte die Eingabeaufforderung (Windows-Taste + R) und geben Sie den Befehl „regsvr32“ gefolgt von dem Namen Ihrer OCX-Datei (Mit vollem Dateipfad!) ein:

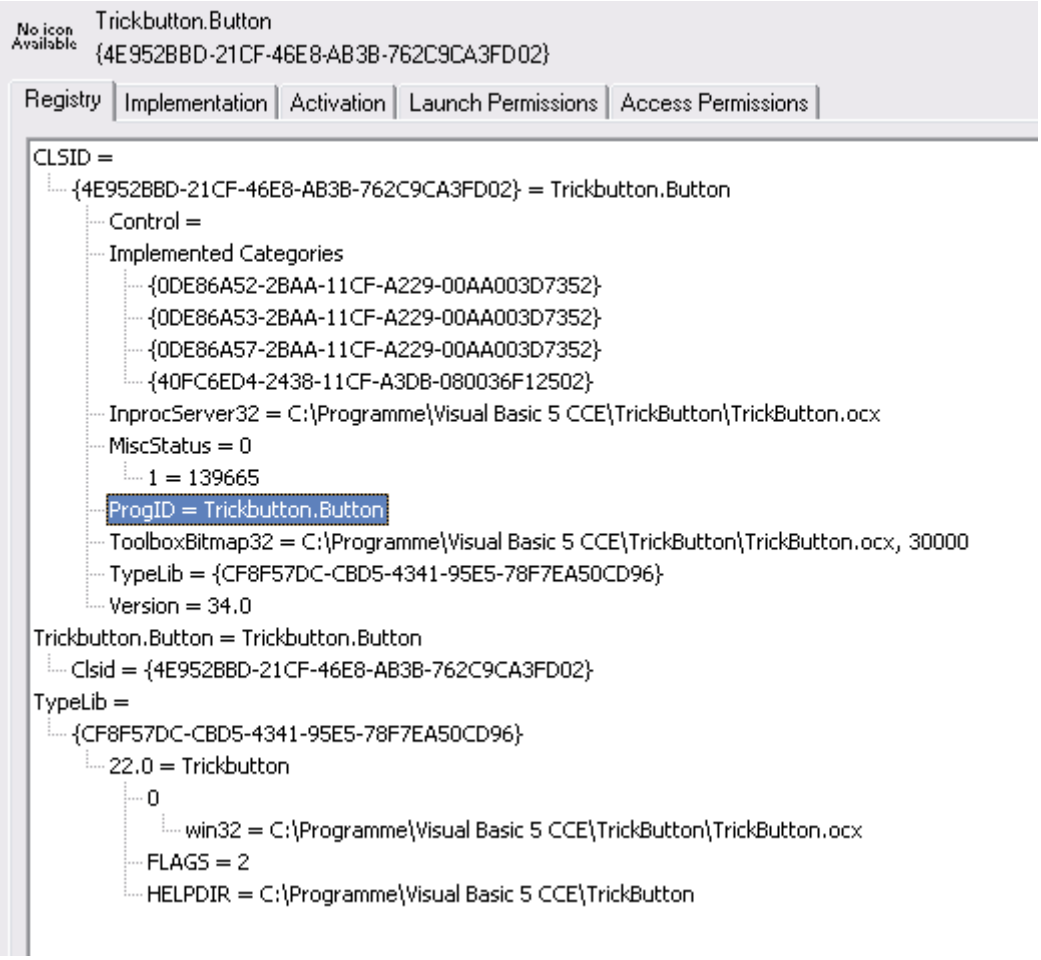


Es erscheint dann eine Meldung, dass die Registrierung durchgeführt wurde:

12. MS OLE/DCOM Ressource Explorer

Beim Einbinden in SAP müssen Sie den Namen des Controls wissen, mit dem das Steuerelement in Windows verwaltet wird. Dabei ist der „Microsoft OLE/ DCOM Ressource Explorer“ behilflich.





Die Datei kann unter
http://www.microsoft.com/downloads/details.aspx?displaylang=en&FamilyID=5233b70d-d9b2-4cb5-aeb6-45664be858b6

Herunter geladen werden.

13. Ideen

13.1. Scannen

Scan Control einbinden, um Dokumente direkt über SAP einzuscannen und im BOR (Business Object Repository) zu speichern.

13.2. Technische Zeichnungen

Laden von Bauplänen eines Maschinentyps, der beim Kunden steht. Auswahl einzelner Elemente um dazu eine Ersatzteillieferung oder einen Serviceauftrag zu erstellen.

13.3. Neues Eingabefeld

In vielen Anwendungen sind Eingabefelder vorhanden, bei denen nach jedem neuen Buchstaben, der eingetippt wird, die Auswahlliste angepasst wird. Das ist auch mit einem entsprechenden Control möglich…

13.4. Messgeräte

Das Einbinden von grafischen Elementen kann es einfacher machen, kritische Werte zu erkennen. Auch wenn Werte ab einer bestimmten Größe mit einem Icon hervorgehoben werden können, um das Erreichen einer Warnschwelle anzeigen kann, kann eine grafische Aufbereitung sinnvoll sein.

13.5. Schieberegler

In SAP können numerische Werte direkt oder über eine DropdownBox eingegeben werden oder über Pushbuttons erhöht und verringert werden. Ein Schieberegler oder Drehrad kann auch hier die Eingabe intuitiver machen.

13.6. Schnittstellen auslesen

Ein OCX-Control könnte auf ein an eine Schnittstelle angeschlossenes Gerät zugreifen, Daten auslesen und im SAP speichern. So könnten z.B. Daten einer Krankenversicherungs-Chipkarte von einem Lesegerät gelesen werden. Das Control liefert die notwendigen Daten zurück, so dass sofort die entsprechende Krankenakte auf dem Bildschirm angezeigt wird.

13.7. Daten per Drag'n'Drop ablegen

Etwas komplizierter, aber sicherlich auch denkbar: ein ActiveX-Control, das Dateien aufnehmen kann, die per Drag and Drop aus dem Explorer auf das Control gezogen werden. So entfiele das mehrmalige Öffnen eines Dateidialogs. Wenn häufig Dokumente zu einem Vorgang abgelegt werden müssen (Fotos, Zertifikate etc. zu einer Qualitätsmeldung), dann kann das viel Arbeit ersparen.

13.8.

14. Ausblick

14.1. Veraltete Technik?

Die OCX-Technik hat ihren den Weg in die aktuellen Entwicklungsumgebungen der .NET Welt nicht mehr gefunden hat, da sie hier durch ein neues Konzept abgelöst wurde.

Ich kann nicht abschätzen, wie lange der SAPGUI noch Bestand haben wird, denn offiziell wurde das Control-basierte User Interface von WebDynpro abgelöst. Hier werden Techniken wie Adobe Flex (FlashIslands) oder Microsoft Silverlight (SilverlightIslands) verwendet, um die Oberfläche zu erweitern.

Dennoch glaube ich, dass der SAPGUI mit seinen Controls noch einige Jahre im SAP Umfeld gute Dienste leisten wird.

15. Weiterführende Links

Umfangreiche Hilfeseite über Visual Basic:

http://www.vbarchiv.net

Erstellung eines OCX-Controls

http://www.vbarchiv.net/workshop/workshop_98-erstellung-eines-kommunikation-ocx.html

Einführung in Visual Basic

http://rhirte.de/vb/home.htm

Visual Basic Tutorials (english)

http://cuinl.tripod.com/tutorials.htm

Die Homepage von Dr. Thomas Müller, der bereits professionelle ActiveX-Controls für das SAP-Umfeld programmiert hat:

http://www.commit-work.com/

Sachwortregister

ActiveX .....*Siehe* OCX
Automation Queue 25, 27
DISPATCH 17
Kurzdump.....28
Object Browser.....29
OCX..... 33
OLE/ DCOM Resource Explorer..... 31
SAPGUI..... 33
Trace 26, 27
VB Projekt 30