Ribbon.dll

Version 0.2.53 vom 25.08.2021

Einführung:

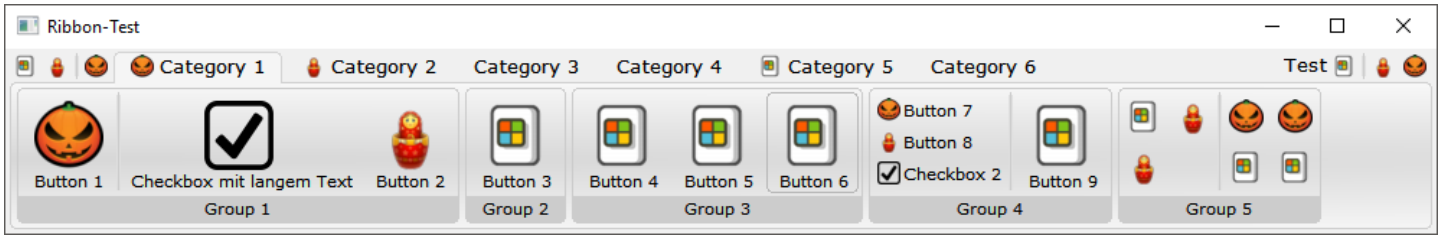
Das Ribbon-Control ist eine Erweiterung der Toolbar. Anders als bei der Toolbar lassen sich hier verschiedene Steuerelemente unterbringen (momentan verschiedene Arten von Buttons, Checkboxen und Text). Diese Elemente lassen sich gruppieren, verschieden anordnen und außerdem in Kategorien einsortieren, damit immer nur relevante Steuerelemente angezeigt werden. Das spart Platz und lässt Programme dadurch übersichtlicher wirken (bzw. es lassen sich mehr Funktionen als in einer Toolbar unterbringen). Highlights:

- Einfache Einbindung in XProfan-Programme dank einer DLL, einer Inc und einer PH-Datei.
- Reduzierung der Funktionen auf das Nötigste, dennoch großer Funktionsumfang für Experten.
- Etliche Funktionen zur Anpassung des Aussehens, von der Hintergrundfarbe bis zur Rundung der Ecken.
- Automatische Anordnung der Steuerelemente (zu keiner Zeit ist ein Eingriff notwendig).
- Kein Wissen über Subclassing oder Windows-APIs für die Grundfunktionen notwendig.
- Automatische Anpassung des Ribbons an die Fensterbreite (falls gewünscht).
- Einfaches System zur Rückgabe von Ereignissen, leicht in die Hauptschleife des Programms integrierbar.
- Erstellung nahezu beliebig vieler Ribbons möglich.

Hinweise für die unfertige Version:

- Die Namen der Funktionen in der DLL können sich noch ändern.
- Funktionen können in kommenden Versionen entfernt werden.
- Das Ribbon funktioniert am besten, wenn die Systemstandardfarbe benutzt wird.
- Die Popup-Menüs sind noch nicht wirklich ausgereift. Radiobuttons und Checkboxen sind besonders betroffen.

Aufbau eines Ribbons:



Die Steuerelemente innerhalb eines Ribbons folgen einer festgelegten Hierarchie:

- Das Ribbon wird erstellt. Dies ist der Ausgangspunkt für alle Steuerelemente.
    - Auf dem Ribbon können *HeadItems* erstellt werden. Das sind *Buttons*, *PushButtons* und *Separatoren* oben links.
    - Es können ebenfalls *RightHeadItems* erstellt werden (*Buttons*, *PushButtons*, *Separatoren*, *Texte* und *TextButtons*), immer rechts im Ribbon dargestellt.
    - Die wichtigsten Items werden aber *Categories* sein. Diese bilden die Grundlage für alle Steuerelemente im Hauptteil des Ribbons. Sie sind vergleichbar mit den Tabs eines Tabcontrols.
        - Alle Steuerelemente innerhalb einer *Category* werden in einer *Group* abgelegt. *Groups* können folgendes enthalten:
            - *Buttons*. Sie geben beim Klicken ein Ereignis zurück.

- *PushButtons*. Wie *Buttons*, nur dass sie beim Klicken zusätzlich ihren Status verändern (normal/gedrückt).
- *Checkboxen*. Sie verhalten sich wie normale Checkboxen.
- *Images*. Das sind Bilder. Sie werden immer auf eine Höhe von 70 Pixeln angepasst und können weder gehovert noch geklickt werden.
- *Separatoren.* Striche von oben nach unten, die Steuerelemente in Gruppen unterteilen.
- *ButtonContainer*. Sie können nur *Buttons* und *PushButtons* sowie *Separatoren* beinhalten (vgl. Group 5). Die *Buttons*/*PushButtons* werden immer übereinander angeordnet (oben -> unten, links -> rechts), bis sie von einem *Separator* unterbrochen werden oder keine weiteren Steuerelemente mehr hinzugefügt wurden. Die Steuerelemente zeigen niemals Text an.
- *Container.* Sie können bis zu 3 frei kombinierbare Steuerelemente enthalten, die übereinander angeordnet sind (vgl. Group 4):
  - *Buttons*
  - *Checkboxen*
  - *PushButtons*

Art des Controls	Parent	Text	Image	Checked
~Ribbon_Type_Button	ButtonContainer/Container/Group	✓/×	✓*	×
~Ribbon_Type_ButtonContainer	Group	×	×	×
~Ribbon_Type_Category	Ribbon	✓	16*16	×
~Ribbon_Type_Checkbox	Container/Group	✓	×	✓
~Ribbon_Type_Container	Group	×	×	×
~Ribbon_Type_Group	Category	✓	×	×
~Ribbon_Type_HeadButton	Ribbon	×	16*16	×
~Ribbon_Type_HeadPushButton	Ribbon	×	16*16	✓
~Ribbon_Type_HeadSeparator	Ribbon	×	×	×
~Ribbon_Type_Image	Group	×	✓**	×
~Ribbon_Type_PushButton	ButtonContainer/Container/Group	✓/×	✓*	✓
~Ribbon_Type_RightHeadButton	Ribbon	×	16*16	×
~Ribbon_Type_RightHeadPushButton	Ribbon	×	16*16	✓
~Ribbon_Type_RightHeadSeparator	Ribbon	×	×	×
~Ribbon_Type_RightHeadText	Ribbon	✓	×	×
~Ribbon_Type_RightHeadTextButton	Ribbon	✓	×	×
~Ribbon_Type_Separator	ButtonContainer/Container/Group	×	×	×
**In Group: 48*48, in Container: 16*16, in ButtonContainer: 24*24*				
**Bildgröße wird verzerrungsfrei auf eine Höhe von 70 Pixel angepasst, Bilder können nicht gehovert oder angeklickt werden.				

Wir erstellen ein Ribbon:

Hier mal ein Überblick, wie man ein einfaches Ribbon erstellt. Eine Erklärung zu den einzelnen Funktionen findet sich im Befehlsindex.
Nach dem Erstellen des Hauptfensters und dem Laden der DLL (und ich gehe davon aus, dass die PH und die Inc eingebunden sind) kann das Ribbon erstellt werden:

```
Declare ribbon&
ribbon&=Create(„Ribbon", %hWnd, ~Ribbon_Color_Auto, ~Ribbon_Color_Auto, ~Ribbon_Style_Black,
~Ribbon_Flag_Autosize | ~Ribbon_Flag_Collapsible)
```

Diese Zeilen Erzeugen ein Ribbon auf dem Hauptfenster. Es benutzt die von Windows eingestellte Hintergrundfarbe. Der Text wird schwarz sein, außerdem passt es sich automatisch an die Fensterbreite an und man kann es einklappen. Nun können wir die einzelnen Steuerelemente nacheinander hinzufügen. Alle Elemente werden mittels **AddRibbonItem()** hinzugefügt (Erklärung der Funktion bitte aus dem Befehlsindex entnehmen):

```
AddRibbonItem(ribbon&, 1, ~Ribbon_Type_Category, "Test1", Create("hIcon", Par$(0), 2), 0)
AddRibbonItem(1, 10, ~Ribbon_Type_Group, "Gruppenname 1", 0, 0)
AddRibbonItem(10, 100, ~Ribbon_Type_PushButton, "Button1", Create("hIcon", Par$(0), 0), ~Ribbon_Status_Checked)
AddRibbonItem(10, 101, ~Ribbon_Type_Separator, "", Create("hIcon", Par$(0), 1), 0)
AddRibbonItem(10, 102, ~Ribbon_Type_Checkbox, "Button2 mit langem Text", Create("hIcon", Par$(0), 2), 0)
AddRibbonItem(10, 103, ~Ribbon_Type_Button, "Button3", Create("hIcon", Par$(0), 3), 0)
```

Wir haben jetzt eine Category mit einer Gruppe und in dieser Gruppe ein paar Steuerelemente. Der PushButton mit der ID 100 ist standardmäßig gepusht (kann durch Click oder über diverse Funktionen geändert werden).
Ein paar Dinge sind zu beachten:

- Jede ID darf nur einmal vergeben werden. Über die ID wird das Control angesprochen. Die ID ist ein Handle, hat aber mit den Handles anderer Controls nichts gemein, da die Verwaltung komplett über die DLL erfolgt.
- Jede ID muss größer als 0 sein. 0 steht für ungültig und IDs kleiner als 0 sind für System-Controls reserviert.
- Die IDs gelten programmweit. Beim Arbeiten mit mehreren Ribbons dürfen die IDs trotzdem nur genau ein Mal vergeben werden! Es ist also nicht möglich ID 100 an Elemente in zwei verschiedenen Ribbons zu vergeben.
- Die DLL löscht niemals Icons oder verändert diese. Icons müssen immer verfügbar sein, solange das zugehörige Control existiert. Der Programmierer ist für das Freigeben des Speichers von Icons nach Gebrauch zuständig (weswegen das obige Beispiel etwas schlampig ist, *hust*).
- Kleine Icons werden gestreckt angezeigt, große entsprechend verkleinert. Trotzdem bleibt das Original-Icon unberührt.
- Die meisten Steuerelemente können nicht alle Eigenschaften anzeigen (Beschriftung, Icon, Status). Trotzdem können sie allen Steuerelementen zugewiesen und aus diesen ausgelesen werden.
- Der Typ eines Steuerelements kann nachträglich nicht geändert werden.
- Die Position der einzelnen Steuerelemente errechnet das Ribbon von selbst. Sie ergibt sich aus der Erstellreihenfolge, kann aber durch den Befehl **MoveRibbonItem()** geändert werden.
- Es ist egal ob Head- und RightHeadItems vor oder nach den Categories und deren Inhalt erstellt werden. Aber auch hier gilt, dass die einzelnen Elemente beider Gruppen von links nach rechts gerendert werden.
- Das Parent eines Steuerelements muss stets vor den Kindern erstellt werden. Das hängt damit zusammen wie die DLL die Ribbons verwaltet.
- Manche Steuerelemente verlangen anstatt einer Parent-ID direkt das Handle des Ribbons. Dieses wird von der Funktion **Create(„Ribbon",…)** zurückgegeben. Die Vorgabe dafür erfolgt durch den Typ des Steuerelements, das erzeugt werden soll.
- Wird ein Steuerelement gelöscht werden auch eventuell vorhandene Substeuerelemente gelöscht.
- XProfan-Standardbefehle wie *SetText, GetText$(), DestroyWindow()* etc. funktionieren auf dem Ribbon, und erst recht auf den Steuerelementen auf dem Ribbon, nicht. Dafür stellt die DLL Befehle bereit.
- Das Ribbon hat im ausgeklappten Zustand immer eine Höhe von 128 Pixeln, im eingeklappten Zustand 24.
- Nach dem Erstellen eines Ribbons und wenn die erste Category erstellt wurde wird sie immer sofort aktiviert. Soll eine andere Category aktiv sein bitte die Funktion **ActivateRibbonCategory()** verwenden.
- Das Ribbon ist nicht für Kinder unter 36 Monaten geeignet, da es verschluckbare Kleinteile enthält!

Nach dieser Methode flackert das Ribbon aber, weil nach jedem neu erzeugten Steuerelement ein Zeichenbefehl kommt und das Ribbon neu gerendert wird. Um das zu verhindern kann man **SetRibbonMetric()** nutzen und das Neuzeichnen verhindern:

```
Declare ribbon&
ribbon&=Create(„Ribbon", %hWnd, ~Ribbon_Color_Auto, ~Ribbon_Color_Auto, ~Ribbon_Style_Black,
~Ribbon_Flag_Autosize | ~Ribbon_Flag_Collapsible)
SetRibbonMetric(ribbon&, ~Ribbon_Metric_UpdateMode, 1)
AddRibbonItem(ribbon&, 1, ~Ribbon_Type_Category, "Test1", Create("hIcon", Par$(0), 2), 0)
AddRibbonItem(1, 10, ~Ribbon_Type_Group, "Gruppenname 1", 0, 0)
AddRibbonItem(10, 100, ~Ribbon_Type_PushButton, "Button1", Create("hIcon", Par$(0), 0), ~Ribbon_Status_Checked)
```

```
AddRibbonItem(10, 101, ~Ribbon_Type_Separator, "", Create("hIcon", Par$(0), 1), 0)
AddRibbonItem(10, 102, ~Ribbon_Type_Checkbox, "Button2 mit langem Text", Create("hIcon", Par$(0), 2), 0)
AddRibbonItem(10, 103, ~Ribbon_Type_Button, "Button3", Create("hIcon", Par$(0), 3), 0)
SetRibbonMetric(ribbon&, ~Ribbon_Metric_UpdateMode, 0)
RenderRibbon(ribbon&)
```

Damit wird das Neuzeichnen verhindert bis der UpdateMode wieder auf normal gestellt wird. Anschließend rendern wir das Ribbon. Wir haben jetzt ein Hauptfenster mit einem Ribbon. Es reagiert bereits auf die Maus und hovert die entsprechenden Steuerelemente. Als nächstes wollen wir auf die Buttons reagieren. Für die normalen Steuerelemente gibt es in XProfan-Programmen üblicherweise eine Schleife, die die auftretenden Ereignisse abfängt und abarbeitet. In einfachster Form:

```
While 1
waitinput
EndWhile
```

Diese Schleife wartet auf Ereignisse. Zwischen waitinput und EndWhile würden die dann abgearbeitet werden (das ist jedem Programmierer bekannt). Um die Ribbon-Ereignisse abzufangen benötigen wir einen Bereich in dem die Ereignisse und dazugehörigen IDs gespeichert werden. Außerdem sendet das Ribbon jedes Mal die Message „10000“ (bzw. „$2710“) an das übergeordnete Fenster (oder eine andere, wenn mit **SetCallbackMessage()** definiert (Diese Funktion ist nur für wirklich sehr erfahrene Programmierer vorgesehen, weil das Senden einer bereits von Windows definierten Message unerwartete Ergebniss erzeugen kann, z.B. sorgt Message 16 dafür, dass das Fenster einen Schließbefehl erhält!)), also möchten wir diese auch noch abfangen:

```
Declare ribbon#
Dim ribbon#, RibbonEvent
UserMessages 10000
While 1
waitinput
If %uMessage=10000
While CheckRibbonActivity(ribbon#)
If ribbon#.event&=~Ribbon_Event_LeftClick And ribbon#.id&=100
MessageBox(„Klick auf 100“, “Klick“, 0)
EndIf
EndWhile
EndIf
EndWhile
```

Es wird:

- …ein Bereich namens ribbon# deklariert und mit der passenden Struktur dimensioniert.
- …dem Programm gesagt, dass die UserMessage „10000“ zum Beenden waitinput führen soll.
- …nach waitinput geprüft ob die UserMessage „10000“ auftrat. Falls ja fanden ein oder sogar mehrere RibbonEvents statt. In diesem Fall…
- …wird geprüft, ob es ein Linksklick war und ob er auf dem Element mit der ID 100 stattfand. Falls ja…
- …wird eine MessageBox ausgegeben.
- Danach wird geprüft, ob weitere Ereignisse anliegen. Deswegen ist **CheckRibbonActivity()** in eine Schleife eingebettet. Damit wird verhindert, dass der Puffer für Ribbon-Ereignisse immer mehr anwächst. Tipp: Mit **FlushRibbonActivity()** kann der Puffer bei Bedarf gelöscht werden.

…und das war’s. Nicht vergessen, dass vorm Beenden des Programms die DLL und der Bereich für die Ribbon-Aktivität sowie eventuell verwendete Icons freigegeben werde sollten. Wir können jetzt einfache Ribbons erstellen und abfragen. Eine Reihe weiterer Funktionen in der DLL geben uns Werkzeuge in die Hand, mit denen wir (wegen Syntax usw. bitte in der Befehlsreferenz nachschauen)…

- …einzelne Steuerelemente aktivieren und deaktivieren können: **DisableRibbonItem()**. Ein deaktiviertes Steuerelement deaktiviert auch immer alle Unterelemente und deren Unterelemente usw. Wenn ich also die Gruppe mit der ID 10 in unserem Beispiel deaktiviere, so werden auch alle Buttons deaktiviert.
- …das Rendern erzwingen: **RenderRibbon()**.
- …die aktuelle Category mittels **GetActiveRibbonCategory()** ermitteln können.

- …die aktuelle Position eines Controls innerhalb seiner Gruppe in der Hierarchie ermitteln oder verändern können: `GetRibbonItemPosition()`/`SetRibbonItemPosition()`.
- …die Schriftart des Ribbons verändern können: `SetRibbonFont()`. Das kann aber zu…interessanten Ergebnissen führen.
- …nachträglich die Texte oder Icons eines Ribbons ändern oder auslesen können: `SetRibbonItemImage()`/`SetRibbonItemText()`, sowie `GetRibbonItemImage()`/`GetRibbonItemText()`.
- …das ganze Ribbon verstecken können: `HideRibbon()`.
- …den Status eines Steuerelementes ändern: `SetRibbonItemStatus()`/`AddRibbonItemStatus()`/`SubRibbonItemStatus()`. Diese Funktion wird noch sehr wichtig werden! Analog dazu lässt sich auch der Status auslesen: `GetRibbonItemStatus()`.
- …den Status des Ribbons ändern oder ermitteln (intern Metric genannt): `SetRibbonMetric()`/`GetRibbonMetric()`.
- …ein Steuerelement entfernen: `RemoveRibbonItem()`.
- …oder das Bild für die Checkboxen durch ein anderes ersetzen: `SetRibbonCheckboxImage()`.

Alles Weitere fällt in die Kategorie…

Erweiterte Ribbon-Programmierung:

Dieses Kapitel eignet sich nur für fortgeschrittene Programmierer, die bereits einige Erfahrung in XProfan haben. In ihm lernen wir wie wir per Subclassing selbst in eingreifen können, wenn das übergeordnete Fenster eines Ribbons in der Größe verändert wird, und wie wir PulldownButtons sinnvoll einsetzen können.

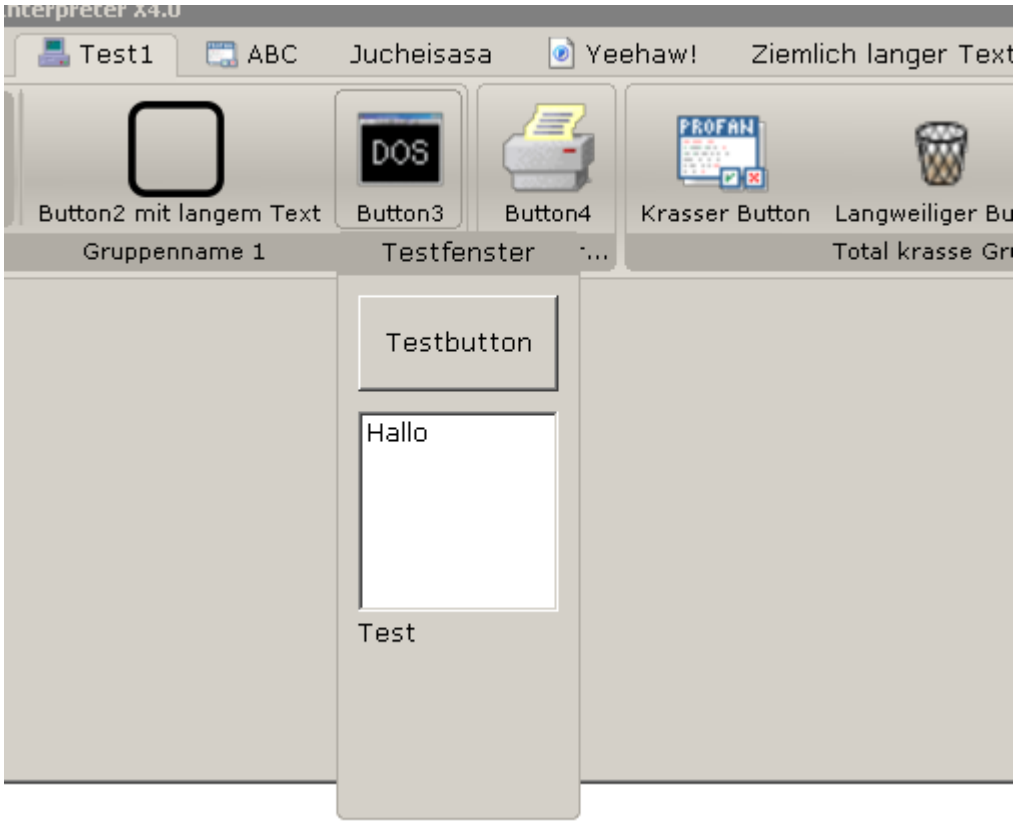
Breitenanpassung selbst übernehmen:
Es kann vorkommen, dass die DLL das Ribbon nicht automatisch anpassen soll, wenn sich die Breite des übergeordneten Fensters ändert. Deshalb hier ein Beispielcode, bei dem sich die SubClassProc darum kümmert (Pfade ggf. anpassen!). Kleiner Tipp: Sollte das Ribbon von vornherein auf einem Fenster erstellt werden, das seine Größe zu keinem Zeitpunkt ändert, so kann das Flag *~Ribbon_Flag_AutoSize* auch weggelassen werden um minimalst Ressourcen zu sparen.

```
$H D:\XProfan\Include\Windows.ph
$H D:\XProfan\Include\Ribbon.ph
$I D:\XProfan\Include\Ribbon.inc

Declare ribbon&, dll&, test$, len&, ribbon#

SubclassProc
If SubclassMessage(%hWnd, ~WM_SIZING) Or SubclassMessage(%hWnd, ~WM_SIZE)
RenderRibbon(ribbon&)
EndIf
Endproc

Windowstyle 31
Window 1200,400
Cls External("user32", "GetSysColor", 15)
dll&=UseDll("D:\XProfan\Include\Ribbon.dll")
ribbon&=Create(„Ribbon", %hWnd, ~Ribbon_Color_Auto, ~Ribbon_Color_Auto, ~Ribbon_Style_Black,
~Ribbon_Flag_Autosize | ~Ribbon_Flag_Collapsible)
SetRibbonMetric(ribbon&, ~Ribbon_Metric_UpdateMode, 1)
AddRibbonItem(ribbon&, 1, ~Ribbon_Type_Category, "Test1", Create("hIcon", Par$(0), 2), 0)
AddRibbonItem(1, 10, ~Ribbon_Type_Group, "Gruppenname 1", 0, 0)
AddRibbonItem(10, 100, ~Ribbon_Type_PushButton, "Button1", Create("hIcon", Par$(0), 0), ~Ribbon_Status_Checked)
AddRibbonItem(10, 101, ~Ribbon_Type_Separator, "", Create("hIcon", Par$(0), 1), 0)
AddRibbonItem(10, 102, ~Ribbon_Type_Checkbox, "Button2 mit langem Text", Create("hIcon", Par$(0), 2), 0)
AddRibbonItem(10, 103, ~Ribbon_Type_PulldownButton, "Button3", Create("hIcon", Par$(0), 3), 0)
SetRibbonMetric(ribbon&, ~Ribbon_Metric_UpdateMode, 0)
RenderRibbon(ribbon&)
Dim ribbon#, RibbonEvent
UserMessages 10000
SubClass %hWnd, 1
SetActiveWindow(%hWnd)
While 1
waitinput
If %uMessage=10000
```

```
While CheckRibbonActivity(ribbon#)
If ribbon#.event&=~Ribbon_Event_LeftClick And ribbon#.id&=100
MessageBox(„Klick auf 100“, “Klick“, 0)
EndIf
EndWhile
EndIf
EndWhile
```

Popups:

Hinweis: Die Popups stehen noch am Anfang ihrer Entwicklung. Es kann sein, dass hier nochmal alles umgekrempelt wird. Die Beschreibung bezieht sich auf den aktuellen Status und wird an die künftigen Versionen angepasst werden!



Was ist ein *PulldownButton*? Ein *PulldownButton* ist ein *Button*, der bei einem Click nicht nur ein Event sendet, sondern auch ein Fenster an seiner Position öffnet (*Popup* gennant). Man kann in dieses Fenster diverse Steuerelemente hineinlegen. Die Abfrage ist etwas kniffelig, deshalb wird die Benutzung ebenfalls nur für erfahrene Programmierer empfohlen (auch hier wird SubClassing verwendet).

Die Fenster schließen sich automatisch, wenn sie den Fokus verlieren, mehr dazu weiter unten. Prinzipiell können beliebige Steuerelemente in das Fenster gelegt werden und so ausgefeilte Unterdialoge erstellt werden (Farbauswahl, Schriftartauswahl, Formatierungsauswahl, Auswahl ohne Ende!).

Erstellt wird ein *PulldownButton* über den normalen Typ *~Ribbon_Type_Button*. Anschließend muss für diesen Button ein Fenster mittel **`LinkRibbonPopup()`** erzeugt werden. Ein Popup kann nachträglich mit **`ResizeRibbonPopup()`** in der Größe geändert werden. Dieses kann anschließend bevölkert werden. Einfache Text lassen sich mit **`AddPopupText()`** erstellen. Hier sollte kein Static verwendet werden, da es Probleme mit der Hintergrundfarbe geben kann (wenn das Ribbon eine andere Hintergrundfarbe als die von Windows vorgegebene hat). Diese Texte können mit **`EditPopupText()`** geändert, **`MovePopupText()`** versetzt und **`RemovePopupText()`** wieder entfernt werden. Andere Steuerelemente (Listboxen, Buttons, Editfelder etc.) werden zuerst auf einem von XProfan erstellten Fenster erstellt (um nicht aufzufallen entweder im nicht sichtbaren Bereich, einem nicht sichtbaren Fenster, einem Fenster mit deaktiviertem Neuzeichnen oder der Größe 0. Das Steuerelement wird anschließend mittels **`MoveControlToRibbonPopup()`** in ein Popup verschoben. Dabei wird automatisch die Schrift an das Ribbon angeglichen.

Ein PulldownButton mit gelinktem Popup öffnet dieses automatisch, wenn der Button gedrückt wird (und gibt ein Event an das XProfan-Programm zurück). Das Popup bleibt so lange geöffnet bis es entweder den Fokus verliert oder es mit **`GetRibbonPopupHandle()`** geschlossen wird. Das ist zu beachten, wenn ein Ereignis verarbeitet wird, bei dem ein Fenster geöffnet wird (z.B. eine MessageBox). In diesem Fall muss anschließend sofort der Fokus an das Popup zurückübertragen werden, mittels **`SetActiveWindow(GetRibbonPopupHandle(Popup_ID))`**.

Steuerelemente, die in das Popup verschoben wurden, lassen sich in XProfan nicht mehr per Clicked() usw. abfragen. Dafür ist Subclassing notwendig. Ein Beispiel mit dem ein Popup für Button 3 erzeugt wird (wie im Bild oben) liegt dem Paket bei (Pfade ggf. anpassen).

Hinweis: Popups funktionieren nicht nur bei Buttons, sondern auch bei den meisten anderen Steuerelementen.