Das Combo-Box-Widget

Version 1.22

Harald Albrecht
Institut für Geometrie und Praktische Mathematik
RWTH Aachen, Bundesrepublik Deutschland
albrecht@igpm.rwth-aachen.de

Mai 1994

Inhaltsverzeichnis

Abbildungsverzeichnis

Kapitel 1

Eine Combo-Box?

1.1 Die wahre Motifation

*Vor langer, langer Zeit gab es einmal eine weltweite Bewegung mit dem (zugegebenermaßen) ungewöhnlichen Namen „comp.windows.x.motif“. Zu jener Zeit waren in dieser Bewegung viele, viele Menschen besessen von einem platz- und zeitfressenden Dämon namens „GUI“, der ihnen vorgaukelte (nicht zu verwechseln mit dem Verb „vorgaucken“), daß nur mit ihm alle Dinge viel einfacher, schneller und vor allem intuitiver erledigt werden könnten. Und weil diese Bewegung nun einmal weltweit war, fand sich alle paar Tage jemand, der immer wieder quengelte, daß es ihm an einer geeigneten Möglichkeit ermangele, darob seine Programme mit dem Benutzer intuitiv in Verbindung treten könnten. Als magisches Zauberwort wiederholten sie dabei immer wieder beschwörend einen aus der Unterwelt der Persönlichen Computer aufgeschnappten Zauberspruch: „Ich brauche eine Combo-Box“. Da beschlossen nun die noch nicht von diesem Dämon besessenen Programmierer, ihren eigenen Zauber zu entwickeln, weil sie mit ihren sagenhaften U**x-Rechnern eine viel mächtigere Magie besaßen als all' diejenigen, die nur über einen verwunschenen PC verfügten.*

Ach ja ... und wenn sie nicht gestorben sind, dann basteln sie noch heute an ihrem Combo-Box-Zauber herum. Denn X11 – so sagen manche Leute – ist in grauer Urzeit verwunschen worden und könne nur dadurch erlöst werden, daß irgendjemand erkennt, wozu es eigentlich da ist und welchen Sinn es hat. Nur ist die Crux daran, daß es in diesem Augenblick sofort verschwinden und durch noch etwas Verwunscheneres und Unbegreiflicheres ersetzt werden wird. Andere Leute behaupten, daß eben dieses bereits passiert sei[1].

Für die noch nicht so des Zauberns unter X11 respektive Motif Kundigen beschreibt die vorliegende Schriftrolle die erforderlichen Zutaten, um sich eine Combo-Box selbst zu zaubern. Benötigt werden dazu aus der Programmküche ein ANSI C-Compiler (tut mir ja leid, aber K&R sind ja nun wirklich reif für die Analen der C-Historie) und die nötige Motivation in Form von Motif-Bibliotheken (1.1 oder 1.2).

Denjenigen, die noch nie eine Combo-Box gesehen, geschweige denn auch nur an soetwas gedacht haben, sei zunächst einmal ein kurzer Abriß zu diesem Thema gewidmet.

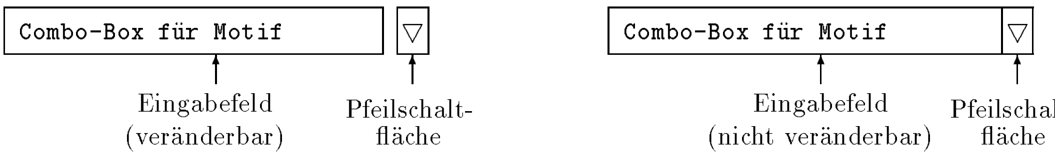

Abbildung 1.1: Eine Combo-Box im Normalzustand

Bekannt wurde die Combo-Box insbesondere durch die Verbreitung der graphischen Oberfläche M$ Windoze[2] auf PCs. Es handelt sich dabei um eine Kombination eines Eingabefeldes mit einer Liste. Diese Liste ist jedoch im Normalfall nicht sichtbar, lediglich ein nach unten weisender Pfeil rechts neben dem Eingabefeld deutet

[1] *Ich entschuldige mich hiermit bei Douglas Adams und natürlich auch bei der OSF...*

[2] *Die in diesem Text genannten Warenzeichen bleiben trotz der verbalen Verunstaltung das Eigentum des Inhabers des betroffenen Warenzeichens.*

darauf hin, daß es sich nicht um ein gewöhnliches Eingabefeld handelt. (Siehe dazu auch die Abbildung 1.1). Die Liste wird erst bei Bedarf „ausgeklappt", indem der Benutzer mit der Maus den nach unten weisenden Pfeil anklickt oder eine bestimmte Tastenkombination betätigt. Daraufhin kann er bequem in der Liste nach dem gewünschten Eintrag suchen. Nach Gebrauch verschwindet die Liste wieder von selbst.

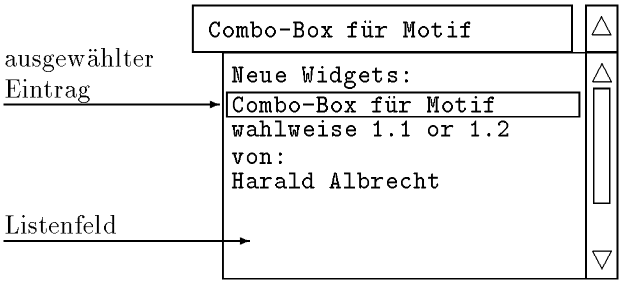

Abbildung 1.2: Eine Combo-Box mit heruntergeklappter Liste

Im Fachjargon von Mikroweich heißt dieser nach unten weisende Pfeil übrigens „Pfeil-Schaltfläche". Weil's so schön scheußlich und damit prägnant ist, verwendet der Autor im weiteren Text auch diesen Begriff. (Im übrigen ist der eMail-Dämon des Autors so programmiert, daß alle elektronische Post mit Beschwerden über die Wahl der Begriffe sofort auf mindestens 4 Terabyte expandiert und an den Absender zurückgeschickt wird. Mehr als 4 Terabyte sind leider bislang wegen des begrenzten Adreßraums nicht möglich.)

1.2 Die Combo-Box aus Benutzersicht

Die Spezies der Combo-Box tritt grundsätzlich in zwei verschiedenen Varianten auf: als Combo-Box mit veränderbarem Eingabefeld und einer ebensolchen mit einem nicht veränderbarem Eingabefeld. Mit welcher der beiden Combo-Boxen es der Benutzer zu tun hat, wird ihm visuell mitgeteilt. Bei einem nicht veränderbarem Eingabefeld schließt sich die Pfeilschaltfläche direkt rechts an das Textfeld an. Hingegen verbleibt bei einer Combo-Box mit veränderbarem Eingabefeld noch ein Zwischenraum von etwa der halben Breite der Pfeilschaltfläche zwischen dem Eingabefeld und eben dieser Pfeilschaltfläche.

Neben der Bedienung mit der Maus bietet die Combo-Box auch noch eine vollständige Tastaturbedienung an. Wie bereits kurz angesprochen, läßt sich das Listenfeld mittels der Tastenkombination $\boxed{\text{META}} + \boxed{\downarrow}$ ausklappen und damit sichtbar machen. Die tatsächliche Taste, die die Funktion der imaginären META-Taste übernimmt, hängt von der jeweiligen Hardware ab. So ist diese Taste beispielsweise bei PC und Silicon Graphics als $\boxed{\text{ALT}}$ bezeichnet, bei Hewlett-Packards 700er als $\boxed{\text{Extended Char}}$ und auf Suns SPARCStation als $\boxed{\diamond}$. So ist halt der systemunabhängige Geist von X11...

Unabhängig davon, ob die Liste nun angezeigt wird oder nicht, kann der Benutzer auch mit den Cursortasten $\boxed{\uparrow}$ und $\boxed{\downarrow}$ die einzelnen Listeneinträge nacheinander abrufen. Bei einer Combo-Box mit veränderlichem Eingabefeld geht dabei der bislang im Eingabefeld stehende Text (auch wenn er bereits verändert wurde) unwiederbringlich verloren und wird durch den Text des Listeneintrags ersetzt.

In Listen mit nicht veränderbarem Eingabefeld kann der Benutzer zusätzlich auch mit Hilfe der Tasten $\boxed{\text{HOME}}$ und $\boxed{\text{END}}$ (als sogenannte virtuelle Tasten von der OSF als „<Key>osfBeginLine" und „<Key>osfEndLine" getauft) schnell an den Anfang oder das Ende der Liste springen. Daneben bietet die Combo-Box in dieser Betriebsart auch noch die sogenannte „Schnellsuche" an. Hierbei springt die Auswahl in der Liste beim Drücken einer (Buchstaben-) Taste automatisch zum nächsten Eintrag, der mit dem eingegebenen Zeichen beginnt. Steht die aktuelle Auswahl bereits auf einem Eintrag, dessen Anfangsbuchstabe mit dem eingegebenen Zeichen identisch ist, so wird die Suche hinter diesem Eintrag fortgesetzt. Damit ist es möglich, nacheinander alle Einträge mit dem gleichen Anfangsbuchstaben anzuwählen.

Von der Intention her ist die Combo-Box die platzsparende Variante einer ordinären Liste, da sie im Gegensatz zu dieser im Normalfall nur den Raum einer einfachen Eingabezeile einnimmt. Trotzdem kann der Benutzer bei Bedarf in der übersichtlicheren und daher auch mehr Platz beanspruchenden Liste suchen.

Das folgende Kapitel gibt zunächst eine Übersicht über die Benutzung des Combo-Box-Widgets. Das daran anschließende Kapitel erläutert dann eingehend die Interna dieses Widgets sowie die diversen Klippen, die es bei der Implementation einer Combo-Box unter X11/Motif zu umschiffen gilt. Insbesondere der auf dem jeweiligen System arbeitende Fenster-Manager wird in den einzelnen Kapiteln immer wieder einmal eine zwielichtige Rolle zu spielen haben.

1.3 Was ist neu an der Version 1.22?

Die augenscheinlichste Änderung gegenüber der allerersten Version 1.0[3] dürfte darin bestehen, daß nun auf Wunsch direkt links neben dem Eingabefeld der Combo-Box auch ein Text oder ein Pixmap erscheinen kann. Dazu bietet die Combo-Box eine Reihe von Ressourcen, die ähnlich denen von XmLabel verwendet werden. Zwecks genauerer Informationen sei der Leser auf die Ressourcenliste im nächsten Kapitel verwiesen.

Weiterhin wurde die „(Vor-) Spiegelung“ von Ressourcen der Kinder-Widgets soweit verbessert, daß nun auch problemlos die Ressourcen in einer Applikations-Ressourcendatei abgelegt sein können. Dabei sind die einzelnen Kinder-Widgets der Combo-Box, aus denen diese zusammengesetzt ist, für den Benutzer nun vollends transparent geworden. Sämtliche Ressourcen – ob sie nun einem Kind der Combo-Box gehören oder nicht – werden sowohl im Programmcode als auch in Ressourcen-Dateien so angesprochen, als ob sie dem Combo-Box-Widget selbst gehören.

Einige kleinere Änderungen betrafen weitere Konfigurationsmöglichkeiten und die Sorge für eine konsistente Farbgebung aller Kinder, sobald eine Farbe am Combo-Box-Widget selbst verändert wird.

Mit der Version 1.2 ist die Combo-Box nun auch Motif-1.2-fest. Diese Version mag nämlich überhaupt keine overrideShellWidgetClass. Statt dessen tut's aber auch eine wmShellWidgetClass genauso gut. Aber darauf muß man ja auch erst kommen. Jedenfalls hat die OSF intern doch einiges zwischen der Version 1.1 und 1.2 verändert.

Neues ist auch bei den Ressourcen zu verzeichnen: ab sofort können **XmNitems** sowie **XmNitemCount** zu jedem beliebigen Zeitpunkt nicht nur abgefragt, sondern nunmehr auch gesetzt werden. Ebenso ist auch die Initialisierung über die Ressourcen-Datei der Applikation möglich.

Außerdem galt es, noch ein paar Speicherlecks in Zusammenhang mit `XmStringGetLtoR` aus der Binärwelt zu räumen (Version 1.21). Die momentan frischeste Version 1.22 bietet endlich auch den Drag'n'Drop-Mechanismus von Motif 1.2. Außerdem arbeitet die Combo-Box nun auch mit dem fvwm-Window-Manager einwandfrei zusammen. Dazu ist die Ressource **XmNtwmHandlingOn** auf `True` zu setzen.

[3] *Wie man sieht, ist der Autor keineswegs abergläubisch und hat daher tatsächlich mit der Versionsnummer 1.0 begonnen, ohne den alten Spruch „Benutze niemals ein Programm vor der Version 1.1“ auch nur eines Gedankens zu würdigen.*

Kapitel 2

Die Combo-Box benutzen

2.1 Konfiguration über Ressourcen

Dreh- und Angelpunkt eines jeden Widgets sind nun einmal seine Ressourcen. Hierüber erfolgt zum einen eine Konfiguration und Parametrisierung, zum anderen die Abfrage des aktuellen Zustands. Im Folgenden sind daher alle zur Verfügung stehenden Ressourcen der Combo-Box aufgeführt.

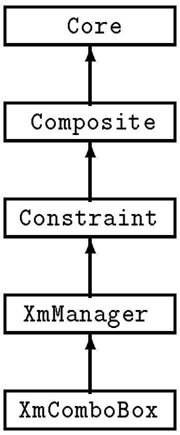

Abbildung 2.1: Stammbaum einer Combo-Box

Diese Dokumentation hält sich dabei an die eingeführten Standards der X11-Literatur, wie sie beispielsweise in den Motif-Handbüchern benutzt werden. Zunächst folgt daher eine Übersicht, in der die einzelnen Ressourcen mit ihrem Namen, Klasse, Type und Voreinstellung tabelliert sind. Daneben gibt diese Übersicht auch darüber Auskunft, ob eine Ressource nur während des Einrichtens eines Widgets (**C** für Create) gesetzt oder aber jederzeit ausgelesen respektive verändert werden darf (**G** für Get und **S** für Set). Daran schließt sich eine genauere Erklärung aller Ressourcen an.

Ein Widget erbt entsprechend seiner Stellung in der Motif-Klassenhierarchie sämtliche Ressourcen seiner Vorfahren. In unserem Fall stammt die Combo-Box indirekt von Composite ab, da sie Kinder enthält und aufgrund des X-Toolkits Xt erst die Widgetklasse Composite sowie deren Nachkommen Kinder haben dürfen. Bei diesen Kindern handelt es sich beispielsweise um ein Eingabefeld, eine Pfeilschaltfläche, eine Liste und noch ein paar andere Widgets.

Um jedoch die durchgängige Tastaturbelegung respektive -bedienung unter Motif auch innerhalb der Combo-Box zu bieten, stammt diese nicht etwa direkt von Composite ab, sondern vielmehr von XmManager. Auf diese Weise führen die Tasten <Key>osfCancel und <Key>osfActivate in Dialogen wie gewohnt zum Abbruch oder zur Bestätigung des Dialogs. Die dafür erforderliche Logik bietet aber erst XmManager. Irgendwelche sonstigen speziellen Eigenschaften von Constraint und XmManager werden hingegen nicht benötigt und daher auch nicht benutzt. Insbesondere erlegt die Combo-Box ihren Kindern keine sogenannten „constraints“ auf.

Nicht aufgeführt in der folgenden Ressourcen-Liste sind deshalb diejenigen ererbten Ressourcen der Elternklassen Core, Composite, Constraint und XmManager, die unverändert übernommen wurden. Trotzdem bietet die Combo-Box getreu der allgemeinen Motif-Philosophie „Angriff ist die beste Verteidigung“ eine hübsche Reihe von immerhin mehr als 30 neuen Ressourcen an, um so möglichst weitgehend konfigurierbar zu sein.

XmComboBox Ressourcen

Name **Klasse**	**Voreinstellung** **Type**	**Zugriff**
XmNalignment XmCAlignment	XmALIGNMENT_CENTER unsigned char	CSG
XmNblinkRate XmCBlinkRate	500 int	CSG
XmNcolumns XmCColumns	dynamisch short	CSG
XmNcursorPosition XmCCursorPosition	0 XmTextPosition	CSG
XmNcursorPositionVisible XmCCursorPositionVisible	True Boolean	CSG
XmNdropDownCallback XmCDropDownCallback	NULL XtCallback	CSG
XmNdropDownCursor XmNDropDownCursor	center_ptr Cursor	CSG
XmNdropDownOffset XmCDropDownOffset	dynamisch Position	CSG
XmNeditable XmCEditable	False Boolean	CG
XmNfontList XmCFontList	dynamisch XmFontList	CSG
XmNitemCount XmCItemCount	0 int	CSG
XmNitems XmCItems	NULL XmStringTable	CSG
XmNlabelFontList XmCLabelFontList	dynamisch XmFontList	CSG
XmNlabelInsensitivePixmap XmCLabelInsensitivePixmap	XmUNSPECIFIED_PIXMAP Pixmap	CSG
XmNlabelMarginBottom XmCLabelMarginBottom	0 Dimension	CSG
XmNlabelMarginHeight XmCLabelMarginHeight	2 Dimension	CSG
XmNlabelMarginLeft XmCLabelMarginLeft	0 Dimension	CSG
XmNlabelMarginRight XmCLabelMarginRight	0 Dimension	CSG
XmNlabelMarginTop XmCLabelMarginTop	0 Dimension	CSG
XmNlabelMarginWidth XmCLabelMarginWidth	2 Dimension	CSG
XmNlabelPixmap XmCLabelPixmap	XmUNSPECIFIED_PIXMAP Pixmap	CSG
XmNlabelString XmCXmString	dynamisch XmString	CSG
XmNlabelType XmCLabelType	XmSTRING unsigned char	CSG
XmNlistMarginHeight XmCListMarginHeight	0 Dimension	CSG
XmNlistMarginWidth XmCListMarginWidth	0 Dimension	CSG

XmComboBox Ressourcen		
Name **Klasse**	**Voreinstellung** **Type**	**Zugriff**
XmNlistSpacing XmCListSpacing	0 Dimension	CSG
XmNmarginHeight XmCMarginHeight	5 Dimension	CSG
XmNmarginWidth XmCMarginWidth	5 Dimension	CSG
XmNmaxLength XmCMaxLength	largest integer int	CSG
XmNpersistentDropDown XmCPersistentDropDown	False Boolean	CSG
XmNselectionCallback XmCCallback	NULL XtCallbackList	CSG
XmNselectThreshold XmCSelectThreshold	5 int	CSG
XmNshowLabel XmCShowLabel	False Boolean	CG
XmNsorted XmCSorted	False Boolean	CSG
XmNstringDirection XmCStringDirection	dynamisch XmStringDirection	CSG
XmNtwmHandlingOn XmCTwmHandlingOn	False Boolean	CSG
XmNtopItemPostion XmCTopItemPostion	1 int	G
XmNvalue XmCValue	"" String	CSG
XmNvisibleItemCount XmCVisibleItemCount	8 Int	CSG

Core Ressourcen		
Name **Klasse**	**Voreinstellung** **Type**	**Zugriff**
XmNborderWidth XmCBorderWidth	0 Dimension	CSG

XmNalignment

Entscheidet über die Ausrichtung des Textes oder der Pixmap, welche(r) links vom Eingabefeld auf Wunsch eingeblendet werden kann. Erlaubte Werte sind `XmALIGNMENT_BEGINNING`, `XmALIGNMENT_CENTER` oder `XmALIGNMENT_END`, die Bedeutung ist die gleiche wie beim XmLabel-Widget.

XmNblinkRate

Spezifiziert die Blinkrate des Cursors in Millisekunden. Dabei gibt diese Zeit jeweils an, wie lange der Cursor sichtbar respektive unsichtbar ist. Ein Wert von 0 für diese Zeit schaltet das Blinken ab.

XmNborderWidth

Dieses ist keine neue Ressource, sondern hier wird nur die Voreinstellung der Elternklasse Core (naja, etwas entferntere Eltern...) überschrieben. Im Normalfall muß daher eine Combo-Box ohne den Rahmen auskommen. Wer's aber dennoch unbedingt benötigt, kann den Wert ja einfach überschreiben.

XmNcolumns

Hierüber kann die Breite des Eingabefeldes – und damit die Gesamtbreite der Combo-Box – festgelegt werden. Zu der Breite des Eingabefeldes kommt natürlich immer noch der von der Pfeil-Schaltfläche belegte Raum hinzu.

XmNcursorPosition

Legt die Position innerhalb des Textes fest, an der sich der Einfügecursor befindet. Dabei wird die Position als Distanz in Zeichen vom Beginn des Textes angegeben – der ersten Zeichenposition entspricht damit der Wert 0.

XmNcursorPositionVisible

Wenn der Einfügeposition durch einen blinkenden Cursor markiert werden soll, ist diese Ressource auf `True` zu setzen.

XmNdropDownCallback

Diese Callbacks werden immer dann aufgerufen, wenn das Listenfeld aus- oder eingeklappt wird. Der Grund `XmCR_SHOW_LIST` zeigt dabei an, daß der Callback just vor dem Moment aufgerufen wird, in dem das Listenfeld auf dem Bildschirm erscheint. Hingegen zeigt `XmCR_HIDE_LIST` an, daß das Listenfeld soeben eingeklappt wurde.

XmNdropDownCursor

Hierüber kann der Cursor festgelegt werden, der immer dann angezeigt wird, wenn sich der Mauszeiger innerhalb der Drop-Down-Liste der Combo-Box befindet.

XmNdropDownOffset

Legt fest, wo sich der linke Rand der Liste relativ zum Eingabefeld befindet. Standardmäßig ist dieser Rand um die Breite der Pfeilschaltfläche nach rechts eingerückt. Es sind hier nur positive Werte erlaubt, negative Werte werden ignoriert.

XmNeditable

Legt fest, ob es sich um eine Combo-Box mit veränderbarem Eingabefeld oder aber um eine solche mit einer statischen Eingabezeile handelt. Im ersten Fall kann der Benutzer nicht nur aus dem Listefeld Einträge auswählen, sondern auch die in der Eingabezeile dargestellte Information beliebig verändern. Im zweiten Fall (**XmNeditable** ist `False`) kann der Benutzer nur Einträge aus dem Listenfeld auswählen, nicht aber verändern.

XmNfontList

Legt die Zeichensatzliste fest, die vom Eingabefeld und dem Listenfeld verwendet werden soll. Wenn hier bei der Initialisierung der Combo-Box der Wert `NULL` eingetragen ist, sucht die Combo-Box bei allen Eltern in der Widgethierarchie nach einem Widget der Klasse XmBulletinBoard, XmVendorShell oder XmMenuShell (beziehungsweise einer Unterklasse davon). Falls ein solches Eltern-Widget gefunden werden kann, übernimmt die Combo-Box kurzerhand dessen Fontliste. Ganz bewußt wird von dieser Ressource die zu verwendenden Fonts für die Anzeige von Text innerhalb des Labels links neben dem Eingabefeld nicht beeinflußt. Dadurch ist es problemlos möglich, für Liste und Eingabefeld gemeinsam und den (Hinweis-) Text links daneben unterschiedliche Schriften einzusetzen.

XmNitemCount

Zeigt an, wieviele Einträge sich momentan im Listenfeld befinden.

XmNitems

Enthält eine Liste von Zeigern, die auf die momentan im Listenfeld eingetragenen Elemente verweisen. Hierüber können somit zu einem späteren Zeitpunkt die Einträge wieder ausgelesen werden. Auf Wunsch können hierüber aber auch auf einen Rutsch alle Einträge gesetzt werden. Damit der Zugriff auf diese Ressource aber einen Effekt zeitigt, muß gleichzeitig auch **XmNitemCount** gesetzt werden, ansonsten ignoriert die Combo-Box das Ansinnen des Programmierers.

XmNlabelFontList

Ähnlich **XmNfontList**, nur dieses Mal speziell für die Anzeige von Text im Textfeld links neben dem Eingabefeld.

XmNlabelInsensitivePixmap

Enthält das Pixmap, das benutzt wird, wenn als **XmNlabelType** der Wert `XmPIXMAP` gewählt und die

Combo-Box momentan insensitiv ist.

XmNlabelPixmap

Anzuzeigendes Pixmap, wenn für **XmNlabelType** der Wert `XmPIXMAP` gewählt wurde.

XmNlabelString

Anzueigender Text, falls **XmNlabelType** auf `XmSTRING` gesetzt wurde.

XmNlabelType

Gibt an, ob ein Text links neben dem Eingabefeld angezeigt werden soll `XmSTRING`, oder aber ein Pixmap `XmPIXMAP`.

XmNlabelMarginBottom

Wie bei XmLabel. Diese Werte werden jedoch nur sehr eingeschränkt ausgewertet, da das Textfeld niemals höher werden kann, als das Eingabefeld der Combo-Box. Deshalb spielt eigentlich nur **XmNlabelMarginLeft** sowie **XmNlabelMarginRight** eine größere Rolle.

XmNlabelMarginHeight

Siehe hierzu die Bemerkungen zu **XmNlabelMarginBottom**.

XmNlabelMarginLeft

Siehe hierzu die Bemerkungen zu **XmNlabelMarginBottom**.

XmNlabelMarginRight

Siehe hierzu die Bemerkungen zu **XmNlabelMarginBottom**.

XmNlabelMarginTop

Siehe hierzu die Bemerkungen zu **XmNlabelMarginBottom**.

XmNlabelMarginWidth

Siehe hierzu die Bemerkungen zu **XmNlabelMarginBottom**.

XmNlistMarginHeight

Legt die Höhe des Zwischenraumes zwischen dem Rand des Listenfeldes und den einzelnen Einträgen fest.

XmNlistMarginWidth

Bestimmt die Breite des Zwischenraumes zwischen dem Rand des Listenfeldes und dem eingegebenen Text.

XmNlistSpacing

Gibt an, wie groß der (vertikale) Zwischenraum zwischen einzelnen Einträgen in der Liste ist. Dieser Wert vergrößert sich noch um den Wert der **XmNhilightThickness**-Ressource in Primitive.

XmNmarginHeight

Spezifiziert den Abstand zwischen dem oberen respektive unteren Rand des Eingabefeldes und den Einträgen im Listenfeld.

XmNmarginWidth

Analog zu **XmNmarginHeight**, nur statt dessen für den Abstand zwischen rechtem (beziehungsweise linkem) Rand des Widgets und dem Text verantwortlich.

XmNmaxLength

Beschränkt die maximale Länge eines über die Tastatur eingegebenen Strings auf den angegebenen Wert. Leider umgeht die Funktion **XmComboBoxSetString** diese Begrenzung, so daß damit auch längere Strings, als diese Begrenzung erlaubt, in das Eingabefeld verfrachtet werden können. An diesem „Gotcha“ ist glücklicherweise aber nicht der Autor schuld, sondern die OSF (siehe dazu auch die Erläuterung zu **XmText(3X)** in den OSF-Handbüchern).

XmNpersistentDropDown

Kommen wir hiermit zu einem der traurigeren Abschnitte im Leben einer X11-Applikation. An dieser Ressource und den damit zusammenhängenden Problemen ist leider die Tastatur-Fokus-Politik des jeweils verwendeten Fenster-Managers schuld. Näheres dazu folgt nachher in einem eigenen Abschnitt.

XmNselectionCallback

Hierüber können die Adressen derjenigen C-Funktionen vermerkt werden, die immer dann aufgerufen werden, sobald ein Eintrag aus dem Listenfeld ausgewählt wurde. Der Grund für das Auslösen des Callbacks ist dabei **XmCR_SINGLE_SELECT**. Damit zeigt die Combo-Box an, daß der Benutzer auch möglicherweise etwas unentschlossen im Listenfeld umherwandert und sich noch nicht für einen bestimmten Eintrag entschließen konnte.

XmNselectThreshold

Legt fest, wieviele Pixel die Maus bei gedrückter Maustaste bewegt werden muß, ehe dieses als Auswahl von Text innerhalb des Eingabefeldes erkannt wird.

XmNshowLabel

Legt fest, ob links neben dem Eingabefeld noch ein Text oder ein Pixmap erscheinen soll. Dieser Parameter kann nur während des Erzeugens einer Combo-Box angegeben werden, danach ist er ohne Wirkung.

XmNsorted

Diese Ressource steuert, ob die Einträge im Listenfeld unsortiert oder sortiert dargestellt werden sollen. Bei einer unsortierten Liste erscheinen die Einträge so, wie sie dem Listfeld vom Programm übergeben wurden. Bei einer sortierten Liste wird hingegen jeder neu hinzugekommene Eintrag automatisch alphabetisch einsortiert. Allerdings werden bei einer Änderung von **XmNsorted** von `False` auf `True` die bereits in der Liste befindlichen Einträge nicht neu sortiert[1]. Also: entweder von vornherein alles sortiert oder bleiben lassen!

XmNstringDirection

Gibt an, in welcher Richtung die Einträge im Listenfeld angezeigt werden (ob von links nach rechts oder aber umgekehrt). Da die ganze Geschichte irgendwie seitens der OSF doch noch nicht so ganz durchdacht scheint, existiert diese Ressource eigentlich nur der Vollständigkeit halber...[2]

XmNtwmHandlingOn

Hiermit kann das seltsame Verhalten des Listenfeldes in Zusammenhang mit bestimmten Window-Managern behoben werden. Sollte sich das Listenfeld immer dann zurückziehen, sobald der Mauszeiger dorthinein bewegt wird, so hilft unter Umständen das Setzen dieser Ressource weiter.

XmNtopItemPosition

Eigentlich mehr der Information dienend zeigt **XmNtopItemPosition** an, welcher Eintrag an oberster Position im Listenfeld angezeigt wird.

XmNvalue

Enthält den Text des Eingabefeldes der Combo-Box (unabhängig davon, ob es sich um eine Combo-Box mit veränderbarem Listenfeld handelt, oder nicht).

XmNvisibleItemCount

Hierüber kann festgelegt werden, wieviele Einträge das Listenfeld jeweils auf einen Schlag anzeigt und somit wird indirekt die Höhe der Liste beeinflußt.

Das noch in der Version 1.0 der Combo-Box vorhandene Problem beim Initialisieren von Ressourcen mittels Applikations-Ressourcendatei ist nun mit der vorliegenden Version 1.1 behoben worden. Sämtliche verfügbaren Ressourcen der Combo-Box erscheinen nach außen hin so, als ob sie dem eigentlichen Combo-Box-Widget gehören und können ohne Probleme auch in einer Ressourcendatei angegeben werden (beispielsweise in Form von „*myApp*`*`*mycombobox*`.columns: 20`“).

2.2 Callback-Informationen

Ein Zeiger auf die folgende Struktur wird dem **XmNdropDownCallback** Callback übergeben.

[1] *Kann mir hier vielleicht jemand einmal erklären, warum die von der OSF von sich aus keine Listenfelder anbieten, die selbst für die Sortierung sorgen? Und wenn ich schon dabei bin, wie wäre es denn mit Listenfeldern, die auch benutzergezeichnete Einträge erlauben, und, und, und... Jajaja – ich weiß ja, daß ich zulange mit M$ Windoze herumgefuhrwerkt habe, aber es gibt doch noch ein paar Dinge, die dort ganz praktisch gelöst waren und die Motif 2.0 gut zu Gesicht stünden.*

[2] *Der Autor dieses Pamphlets konnte jedenfalls bislang noch nicht herausfinden, ob auch Text vertikal von oben nach unten beziehungsweise umgekehrt von Motif unterstützt wird. Er tendiert jedoch zu der Annahme, daß hierzu den Programmierern bei der OSF einfach die* Motif*ation fehlte.*

```
typedef struct {
    int      reason;    /* Grund                  */
    XEvent   *event;    /* Ausloesender Event     */
} XmComboBoxDropDownCallbackStruct;
```

Dabei zeigt `reason` an, aus welchem Grund der Callback aktiviert wurde. Hier kommt als Grund entweder `XmCR_SHOW_LIST` oder `XmCR_HIDE_LIST` in Betracht, womit angezeigt wird, daß die Liste ein- oder ausgeklappt wurde.

reason

Zeigt an, warum dieser Callback aufgerufen wurde.

event

Zeigt auf den `XEvent`, welcher den Callback auslöste. Hier kann auch `NULL` stehen, wenn die Liste aufgrund eines Aufrufs von `XmComboBoxShowList` oder `XmComboBoxHideList` ein- oder ausgeklappt wurde.

Ein Zeiger auf die folgende Struktur wird dem **XmNselectionCallback** Callback übergeben.

```
typedef struct {
    int      reason;    /* Grund                  */
    XEvent   *event;    /* Ausloesender Event     */
    XmString value;     /* Ausgewaehlter Eintrag  */
    int      index;     /* Index dieses Eintrages */
} XmComboBoxSelectionCallbackStruct;
```

Dabei zeigt `reason` an, aus welchem Grund der Callback aktiviert wurde. Bislang existiert dafür nur `XmCR_SINGLE_SELECT`, womit angezeigt wird, daß der Benutzer einen (neuen) Eintrag in der Liste der Combo-Box anwählte.

reason

Zeigt an, warum dieser Callback aufgerufen wurde.

event

Zeigt auf den `XEvent`, welcher den Callback auslöste.

value

Zeigt auf einen XmString, der den Text des angewählten Eintrags enthält. Hier sollte eigentlich klar sein, daß der aufgerufene Callback diesen String nicht freigeben oder manipulieren darf!

index

Index (von 1 beginnend) des angewählten Eintrags in der Liste.

2.3 Translationen

Die XmComboBox enthält die Translationen von XmManager. Zusätzlich werden bei der Eingabezeile (einem XmTextField-Widget) je nach Konfiguration einige Translationen überschrieben beziehungsweise hinzugefügt.

<Key>osfDown:	**ComboBox-Manager(down)**
<Key>osfUp:	**ComboBox-Manager(up)**
Alt<Key>osfDown:	**ComboBox-Manager(show-hide-list)**
Meta<Key>osfDown:	**ComboBox-Manager(show-hide-list)**
Alt<Key>osfUp:	**ComboBox-Manager(hide-list)**
Meta<Key>osfUp:	**ComboBox-Manager(hide-list)**
<Key<osfCancel:	**ComboBox-Manager(cancel)**
<Key>Return:	**ComboBox-Manager(activate) activate()**

Speziell bei der Combo-Box mit nicht editierbarem Eingabefeld kommen noch weitere Translationen hinzu.

<Key>osfBeginLine:	**ComboBox-Manager(top)**
<Key>osfEndLine:	**ComboBox-Manager(bottom)**

Das Listenfeld bekommt keine neuen Translationen – hier kann alles so bleiben, wie es ist. Und das ist bei der Combo-Box auch der Trick: Tastatureingaben erhält *nur* das Eingabefeld, nicht aber die Liste. Die Liste nimmt somit nur Mauseingaben entgegen, mit der Tastatur hat sie, da sie einfach in eine OverrideShell verpackt wurde, nichts mehr zu tun. Um es noch einmal deutlich zu sagen: Die hier aufgeführten Translationen werden beim Eingabefeld, also einem Kind der Combo-Box, hinzugefügt. Sollten hier wirklich einmal Änderungen erforderlich werden, so stehen die Funktionen `XmComboBoxGetEditCtrl()` sowie noch `XmComboBoxGetListCtrl()` und `XmComboBoxGetLabelCtrl()` zur Verfügung. Allerdings wird von der Benutzung dieser Funktionen dringends abgeraten, da sie Kenntnisse über den nach außen hin unsichtbaren inneren Aufbau der Combo-Box voraussetzen und damit die Abstraktion, die das Combo-Box-Widget von Hause aus bietet, unterlaufen.

Aber nichts ist so vollkommen... am allerwenigsten Sun. Hier kann es zu Problemen kommen, wenn ein Anwender in einer Combo-Box den Drag'n'Drop-Mechanismus startet. Sollte dieses der Fall sein, so kompilieren Sie den Quellcode der Combo-Box mit dem Schalter `-DNODRAGNDROP` erneut. Dadurch wird Drag'n'Drop in jeder Combo-Box deaktiviert, indem dem Listen sowie dem Eingabefeld neue Translationen hinzugefügt werden.

<Btn2Down>:	**ComboBox-Manager(no-operation)**

Ansonsten funktioniert Drag'n'Drop mit Motif 1.2.3 prächtig. Die vorhergehende Version 1.21 der Combo-Box hatte hingegen noch Problem damit, die daraus resultierten, daß das Listenfeld in einem wmShellWidget steckte. Benutzt man statt dessen ein vendorShellWidget, so funktioniert alles einwandfrei. Dummerweise bin ich nicht eher darauf gekommen...

2.4 Action-Routinen

XmComboBox besitzt die folgenden Action-Routinen. (Naja, soviel Action ist hier ja auch wider nicht...)

ComboBox-Manager(*operation***)**

Der Parameter *operation* gibt an, was zu tun ist. Hierbei bewegt „up" die Auswahl im Listenfeld um einen Eintrag nach oben, „down" die Auswahl in die entgegengesetzte Richtung nach unten. Über „show-hide-list" wird das Listenfeld aus- oder wieder eingeklappt, wohingegen „hide-list" grundsätzlich die Liste einklappt. Daneben signalisiert ein „activate", daß der Benutzer die Eingabetaste drückte, um einen Eintrag zu übernehmen und die Liste zu schließen. Schließlich gibt es noch „top" und „bottom", die den ersten oder den letzten Eintrag auswählen. Last but not least sorgt „cancel" dafür, daß eine eventuell ausgeklappte Liste wieder eingeklappt wird. Ist die Liste bereits eingezogen worden, so wird die ESC-Taste zwecks weiterer Verarbeitung an das Vater-Widget weitergeleitet. Damit ist es dann in Dialogen möglich, diese jederzeit abzubrechen. Als wirklich allerletzte Operation verbleibt jetzt noch „no-operation", die, wie der Name schon andeutet, eben nichts tut.

2.5 Schnittstellenfunktionen

Nachdem im vorangegangenen Abschnitt die Konfiguration mittels Ressourcen beschrieben wurde, folgt nun eine Erläuterung aller Schnittstellenfunktionen. Über diese Funktionen lassen sich bequem die in der Combo-Box dargestellten Informationen abfragen beziehungsweise verändern.

Dabei handelt es sich im Groben um diejenigen Funktionen, die Motif bereits für die Manipulation von XmTextField- und XmList-Widgets zur Verfügung stellt. Nur sind jetzt die Silben „XmTextField" beziehungsweise „XmList" der neuen Silbe „XmComboBox" gewichen. Nichtsdestotrotz sollte aber niemals direkt auf die einzelnen Widgets der Combo-Box zugegriffen werden, sondern nur über die dazu bereitgestellten Funktionen, da diese zum einen teilweise eine erweiterte Funktionalität bieten und zum anderen dafür sorgen, daß die Combo-Box zu jedem Zeitpunkt weiß, in welchem Zustand sie sich befindet.

XmComboBox-Funktion	**siehe auch:**
`void XmComboBoxAddItem(Widget w, XmString item, int pos)` Einen Eintrag (aka: „String“) dem Listenfeld einverleiben. Ist `XmNsorted` auf `True` gesetzt, so wird der Parameter `pos` ignoriert und der übergebene String entsprechend einsortiert. Bei unsortierten Listenfeldern gelten die gleichen Annahmen wie bei `XmListAddItems`.	`XmListAddItem`
`void XmComboBoxAddItems(Widget w, XmString *items, int item_count, int pos)`	`XmListAddItems`
`void XmComboBoxAddItemUnselected(Widget w, XmString item, int pos)`	`XmListAddItemUnselected`
`void XmComboBoxDeleteItem(Widget w, XmString item)`	`XmListDeleteItem`
`void XmComboBoxDeleteItems(Widget w, XmString *items, int item_count)`	`XmListDeleteItems`
`void XmComboBoxDeletePos(Widget w, int pos)`	`XmListDeletePos`
`void XmComboBoxDeleteItemsPos(Widget w, int item_count, int pos)`	`XmListDeleteItemsPos`
`void XmComboBoxDeleteAllItems(Widget w)`	`XmListDeleteAllItems`
`void XmComboBoxReplaceItems(Widget w, XmString *old_items, int item_count, XmString *new_items)`	`XmListReplaceItems`
`void XmComboBoxReplaceItemsPos(Widget w, XmString *new_items, int item_count, int position)`	`XmListReplaceItemsPos`
`Boolean XmComboBoxItemExists(Widget w, XmString item)`	`XmListItemExists`
`int XmComboBoxItemPos(Widget w, XmString item)`	`XmListItemPos`
`Boolean XmComboBoxGetMatchPos(Widget w, XmString item, int **pos_list, int *pos_count)`	`XmListGetMatchPos`
`void XmComboBoxSelectPos(Widget w, int pos, Boolean notify)`	`XmListSelectPos`
`int XmComboBoxGetSelectedPos(Widget w)` Diese Routine arbeitet etwas anders als `XmListGetSelectedPos`, da es in der Liste einer Combo-Box höchstens immer nur einen markierten Eintrag geben kann. Ist kein Eintrag markiert, so liefert die Funktion 0 zurück, ansonsten den Index des momentan angewählten Eintrags.	—
`void XmComboBoxClearSelection(Widget w, Time time)`	`XmTextFieldClearSelection`
`Boolean XmComboBoxCopy(Widget w, Time time)`	`XmTextFieldCopy`
`Boolean XmComboBoxCut(Widget w, Time time)`	`XmTextFieldCut`
`XmTextPosition XmComboBoxGetInsertionPosition( Widget w)`	`XmTextFieldGetInsertionPosition`
`XmTextPosition XmComboBoxGetLastPosition(Widget w)`	`XmTextFieldGetLastPosition`
`int XmComboBoxGetMaxLength(Widget w)`	`XmTextFieldGetMaxLength`
`char *XmComboBoxGetSelection(Widget w)`	`XmTextFieldGetSelection`
`Boolean XmComboBoxGetSelectionPosition(Widget w, XmTextPosition *left, XmTextPosition *right)`	`XmTextFieldGetSelectionPosition`
`char *XmComboBoxGetString(Widget w)`	`XmTextFieldGetString`
`void XmComboBoxInsert(Widget w, XmTextPosition position, char *value)`	`XmTextFieldInsert`
`Boolean XmComboBoxPaste(Widget w)`	`XmTextFieldPaste`
`Boolean XmComboBoxFieldRemove(Widget w)`	`XmTextFieldRemove`

XmComboBox-Funktion	siehe auch:
`void XmComboBoxFieldReplace(Widget w, XmTextPosition from_pos, XmTextPosition to_pos, char *value)`	`XmTextFieldReplace`
`void XmComboBoxSetAddMode(Widget w, Boolean state)`	`XmTextFieldSetAddMode`
`void XmComboBoxSetHilight(Widget w, XmTextPosition left, XmTextPosition right, XmHighlightMode mode)`	`XmTextFieldSetHilight`
`void XmComboBoxSetInsertionPosition(Widget w, XmTextPosition position)`	`XmTextFieldSetInsertionPosition`
`void XmComboBoxSetMaxLength(Widget w, int max_length)`	`XmTextFieldSetMaxLength`
`void XmComboBoxSetSelection(Widget w, XmTextPosition first, XmTextPosition last, Time time)`	`XmTextFieldSetSelection`
`void XmComboBoxSelectItem(Widget w, XmString item, Boolean notify)`	`XmListSelectItem`
`void XmComboBoxSelectPos(Widget w, int pos, Boolean notify)`	`XmListSelectPos`
`void XmComboBoxSetString(Widget w, char *value)` Zumindest Motif 1.1 enthält noch einen bösen Fehler, wenn man versucht, einen leeren String (“”) in das Eingabefeld hineinzuschreiben. Er wird von dieser Routine jedoch entsprechend umschifft.	`XmTextFieldSetString`
`void XmComboBoxShowPosition(Widget w, XmTextPosition position)`	`XmTextFieldShowPosition`
`void XmComboBoxShowList(Widget w)` Klappt die Liste aus.	—
`void XmComboBoxHideList(Widget w)` Klappt die Liste ein.	—

2.6 Die Sache mit dem Tastaturfokus

Kommen wir damit zu einem heikleren Thema. Beläßt man die Ressource `XmNpersistentDropDown` in ihrer Voreinstellung (`False`), so kann der folgende, unter Umständen störende, Effekt beobachtet werden.

Klappt man (beispielsweise mit der Maus) das Listenfeld der Combo-Box herunter und bewegt dann den Mauszeiger (irrtümlich) aus dem Fenster heraus, in dem sich die Combo-Box befindet, so verliert diese den Tastaturfokus. Damit zieht sich das Listenfeld aber auch völlig erschreckt zurück. Dieses Verhalten ist nicht der Combo-Box selbst anzukreiden, sondern liegt im Prinzip begraben, wie X11 den Tastaturfokus behandelt. Dabei ist es durchaus üblich, wenn der Fenstermanager entsprechend konfiguriert ist, daß jeweils das Fenster unter dem Mauszeiger den Tastaturfokus bekommt. Ist hingegen der Fenstermanager auf den sogenannten “click-to-type“-Modus eingestellt, so muß der Benutzer ein Fenster erst einmal anklicken, ehe dieses den Tastaturfokus erhält und damit die Tasteneingaben annimmt[3].

Stört das für den Benutzer zuerst unerwartete „Verschwinden“ des Listenfelds, so kann versucht werden, dieses Einklappen der Liste zu umgehen. Dazu ist `XmNpersistentDropDown` auf `True` zu setzen (beispielsweise in den Applikations-Ressourcen des jeweiligen Programms). Bislang kann es jedoch je nach Fenstermanager dabei zur Verwirrung des Listenfeldes kommen, so daß sich dieses eine zeitlang überhaupt nicht mehr ausklappen läßt.

Bliebe als Fazit: gegebenenfalls einfach einmal ausprobieren und nicht schimpfen (schon gar nicht auf den Autor!), wenn's dann doch nicht geht.

Jedenfalls funktioniert dieser Mechanismus zumindest bei der Maschine, auf der der Autor die Combo-Box entwickelt hat: eine schnuckelige SGI Indigo mit dem 4Dwm (mwm-Ableger) als Fenstermanager. Ebenso

[3] *Der Autor tröstet sich hierbei damit, daß auch die Programmierer, die M$ Windoze fabrizierten, diesbezüglich einige böse Tricks im Windows-Programmcode unterbringen mußten. Ansonsten wäre es dort nicht möglich, den Dialog zu verschieben, in dem sich die Combo-Box befindet, da dann ein ausgeklapptes Listenfeld einfach an seiner alten Position auf dem Bildschirm verharrte – ziemlich irritierend für einen Benutzer.*

(natürlich) auch auf der privaten „LinuxBox“ mit fvwm als Fenstermanager. Bei letzterem Fenstermanager muß **XmNtwmHandlingOn** auf `True` gesetzt werden (am besten in irgendeiner Ressourcendatei diese Einstellung einmal „abladen“).

Bleibt noch ein Nachtrag: die aktuelle Version 1.22 besitzt einen relativ aufwendigen Mechanismus, um mit möglichst vielen Window-Managern zurechtzukommen. Leider gibt es allerdings einige Window-Manager, deren Verhalten derart konfus ist, daß der Benutzer die Combo-Box darauf vorwarnen muß. Hierzu dient insbesondere die neue Ressource `XmNtwmHandlingOn`. Siehe hierzu die Übersicht über alle Ressourcen.

2.7 Einrichten einer Combo-Box

Der programmtechnische Einsatz der Combo-Box ist recht einfach. Die Combo-Box wird einfach mit einer der unzähligen Xt-Funktionen wie beispielsweise `XtCreateManagedWidget`, `XtVaCreateManagedWidget` eingerichtet.

Der an diese Funktionen zu übergebende Klassenname lautet dabei `xmComboBoxWidgetClass`. Eine besondere „Convenience“-Funktion existiert nicht, sie ist auch wirklich nicht nötig.

Irgendwo in Ihrem Programmcode sollte daher so etwas wie das folgende Programmfragment erscheinen:

```
#include <ComboBox.h>

Widget ComboBox, Parent;

ComboBox = XtVaCreateManagedWidget(
    "combobox", xmComboBoxWidgetClass, Parent,
    XmNeditable, True oder False,
    weitere Ressourcen...
    NULL);
```

Das Setzen der `XmNeditable` Ressource ist nicht unbedingt notwendig, aber es ist doch ratsamer, dieses gleich hier zu tun, als beispielsweise in der Applikations-Ressourcen-Datei (es sei denn, Sie möchten es den Benutzern erlauben, Chaos zu stiften).

Danach kann dann die Combo-Box mit Hilfe der bereits vorgestellten Funktionen mit Inhalt gefüllt werden. In diesem Zusammenhang sei daher insbesondere auf die Funktion `XmComboBoxAddItem` verwiesen. Den ausgewählten Text (respektive neu eingegebenen Text) erfahren Sie dann mittels `XmComboBoxGetString`.

Soll dann noch ein bestimmter Eintrag als Voreinstellung gesetzt werden, so ist dazu bei einer nicht editierbaren Combo-Box einer der beiden folgenden Aufrufe auszuführen:

```
XmComboBoxSelectPos(ComboBox, Nr. des Eintrags, False);
```

oder wahlweise:

```
XmComboBoxSelectItem(ComboBox, itemString, False);
```

Hierbei bedeutet jeweils der letzte Parameter, daß der **XmNselectionCallback**-Callback nicht aufgerufen wird. Bei **XmComboBoxSelectPos** ist als zweiter Parameter dabei der Index des gewünschten Eintrags anzugeben, während bei **XmComboBoxSelectItem** hier statt dessen ein **XmString** erwartet wird, der den Text des anzuwählenden Eintrags enthält.

Bei einer Combo-Box mit veränderbarem Eingabefeld kann hingegen auch noch ein anderer Weg beschritten werden, insbesondere dann, wenn ein Text gesetzt werden soll, der nicht in der Liste vorhanden ist.

```
XmComboBoxSetString(ComboBox, "Mein Text");
```

Da der Autor partout keinen Platz schinden möchte – denn schließlich ist die Dokumentation für ein einzelnes Widget ja schon reichlich umfangreich geworden – sei ansonsten auf das zusammen mit den restlichen Sourcen der Combo-Box mitgelieferte Beispielprogramm namens `ComboBoxDemo.c` verwiesen.

2.8 Wo's bisher schon läuft

Hier eine Liste aller Konfigurationen, auf denen das Combo-Box-Widget bislang seinen Dienst verrichtet.

DEC	DECstation 5000/240: ULTRIX 4.3 mit Motif 1.1
	VAX: VMS5.5-2 mit Motif 1.1
Hewlett-Packard	700er mit HPUX 9.01 mit Motif 1.2
IBM	RS6000: AIX 3.2 mit Motif 1.1, 1.2
PC	SCO Unix mit Motif 1.1
	Linux (aber natürlich!) mit Motif 1.2
Silicon Graphics	IRIS Indigo: IRIX 4.0.5 mit Motif 1.1
Sun	SparcStation: SunOS 4.1.3 mit Motif 1.1 und Motif 1.2

Momentan ist die Liste noch sehr übersichtlich – aber das sollte sich mit der Zeit noch ändern. Da der Quelltext der Combo-Box unter den Bedingungen der „GNU General Public License“[4] jederman zugänglich ist, steht somit dem Einsatz auf anderen Systemen nichts im Wege.

In diesem Zusammenhang sei daher insbesondere auf den feinen Unterschied zwischen der sogenannten „offenen“ Software der OSF und der „freien“ Software hingewiesen: Letztere ist in der Tat so offen, daß jeder kostenlos oder gegen eine geringe Bearbeitungsgebühr nicht nur die Software, sondern zugleich mit dem Quellcode auch noch das Know-How bekommt. Bei der „Open Software Foundation“ hingegen erhält man erst Einblick nach Entrichtung eines nicht eben geringen Obulus. Irgendwie muß ich wohl bislang unter „offen“ etwas völlig Falsches verstanden haben – mit Offenheit hat es anscheinend nichts zu tun...

[4] *Siehe hierzu die Datei* `COPYING`*, die zusammen mit den restlichen Dateien geliefert wurde.*

Kapitel 3

Hinter den Kulissen

Nach der vielen Theorie geht es nun schnurstracks an die Bits und Bytes: Motif in der Programmierpraxis. Leider gilt aber insbesondere hier der Satz von der Divergenz zwischen Papier und programmtechnischer Praxis. Und dieses um so mehr, je komplexer das behandelte Produkt ist.

Im folgenden seien nun die Interna und Programmiertricks (respektive -kniffe) vorgestellt, die eine XmComboBox zum Leben erwecken.

3.1 Der Grüne Punkt

Damit zuerst die wirklich gute Nachricht: obwohl die Combo-Box nun wirklich zum allergrößten Teil aus wiederverwerteten Widgets der Motif-Bastelkiste besteht, müssen Sie, verehrter Leser, doch keinen Pfennig (zusätzlich) dafür berappen. Glücklicherweise sind Quellcodes bislang noch vom DSD (Duales System Deutschland) nicht als wert-sch(r)öpf-fähig erkannt worden, weshalb der Autor noch nicht gezwungen ist, überall den grünen Punkt „draufzubappen".

Und weil nach guter deutscher Sitte gefälligst draufzustehen hat, was denn nun tatsächlich drin ist, zuerst einen Blick in den Aufbau eines Combo-Box-Widgets. Wie bereits angesprochen, baut ein guter Teil der Funktionalität einer Combo-Box auf bereits vorhandenen Elementen der Motif-Oberfläche auf. Figur 3.1 zeigt, welche Widgets von welchen anderen Widgets innerhalb der Combo-Box-Hierarchie verwaltet werden.

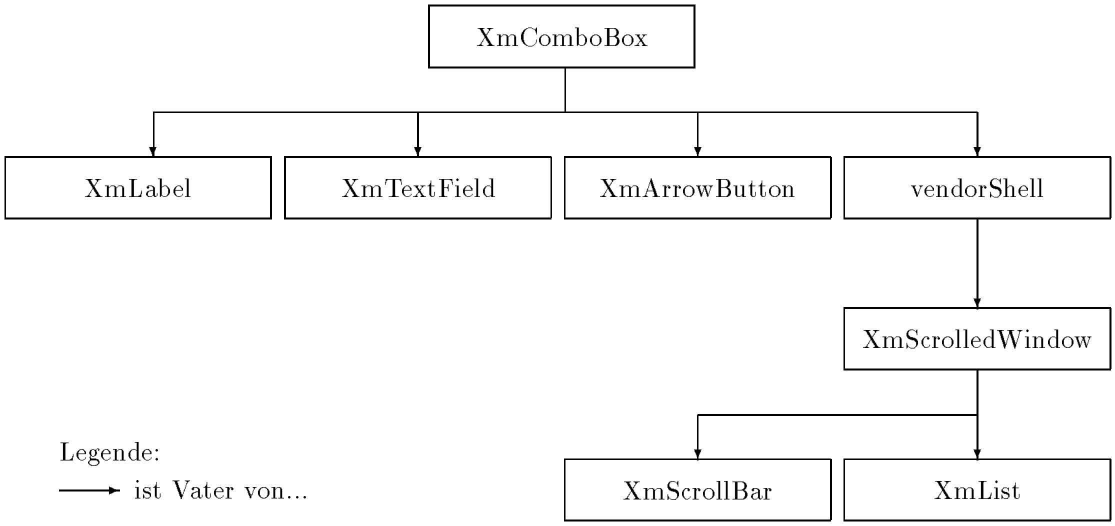

Abbildung 3.1: Widgethierachie innerhalb einer XmComboBox

Nach außen hin tritt die Combo-Box sowohl für den Benutzer als auch für den Programmierer als ein monolithischer Block auf. Ressourcen werden grundsätzlich nur bei der Combo-Box verändert – niemals darf direkt auf Ressourcen der Kinderwidgets zugegriffen werden. Nur auf diese Weise bleibt sichergestellt, daß die Combo-Box zu jedem Zeitpunkt die vollständige Kontrolle über den Zustand aller Kinder behält. Verstöße gegen diese Regel dürften in der Regel früher oder später mit einem seltsamen Verhalten der Combo-Box bis hin zu einem satten Core-Dump geahndet werden.

3.2 Die Routinen im Einzelnen

Im Folgenden seien nun all' diejenigen Routinen vorgestellt, die für die Funktion der Combo-Box (und ebenso für Composite-Widgets) eine Bedeutung haben. Auf die vielen Interface-Funktionen, mit denen Daten in die Combo-Box hinein und wieder aus ihr heraus transferiert werden können, sei hier jedoch nicht weiter eingegangen. Sie sind allgemein so kurz und überschaubar, daß eine explizite Erwähnung unnötig erscheint. Der interessierte Leser sei hierfür auf den Quellcode verwiesen. Dieser Abschnitt soll hingegen grundsätzliche Gedankengänge zur Funktionsweise der Combo-Box aufzeigen.

```
void DefaultGeometry(XmComboBoxWidget w, Dimension *TotalWidth, Dimension *TotalHeight,
Dimension *EditCtrlWidth)
```

In der Funktion `DefaultGeometry()` wird der Raum ermittelt, den die Combo-Box auf dem Bildschirm einnehmen muß, damit das Eingabefeld sowie die Pfeil-Schaltfläche und ein eventueller Text darin Platz finden. Je nachdem, ob es sich um eine Combo-Box mit veränderbarem oder einem statischen Eingabefeld handelt, wird zwischen dem Eingabefeld und dem Pfeil ein zusätzlicher Raum belassen. Dieser hat in etwa die halbe Breite des Pfeils.

`DefaultGeometry()` wird als Hilfsroutine von der `QueryGeometry()`-Methode und während der Initialisierung aufgerufen. Die Initialisierungsroutine `Initialize()`, die eine einzelne Widgetinstanz einrichtet (sozusagen betriebsfertig herrichtet), kann damit eine Voreinstellung für die Widgetgröße ermitteln.

```
void DoLayout(XmComboBoxWidget w)
```

Anscheinend hat sich, jedenfalls was die Namensgebung dieser Methode angeht, ein ungeschriebenes Gesetz herausgebildet, der sich der Autor natürlich nicht verschließen will. Und so heißt auch bei diesem Widget diejenige Funktion wieder `DoLayout()`, die für die Anordung der Kinderwidgets verantwortlich zeichnet. Bei `DoLayout()` handelt es sich um eine der sogenannten „Methoden“, die als Reaktion auf bestimmte Ereignisse oder dergleichen vom Xt-Toolkit aufgerufen werden.

Dabei ordnet `DoLayout()` das Eingabefeld jeweils links an und verbreitert es soweit, daß eben noch der Platz für den Pfeil übrigbleibt. Ein noch eventuell am linken Rand anzuzeigender Text wird dabei entsprechend mit berücksichtigt. Sollte sich die Größe der Combo-Box just dann verändern, wenn die Liste ausgeklappt ist, so wird automatisch diese ebenfalls in ihrer Breite entsprechend den neuen Dimensionen angepaßt. Diese Aufgabe übernimmt dabei freundlicherweise die nächste zu beschreibene Funktion: `DoDropDownLayout()`.

```
void DoDropDownLayout(XmComboBoxWidget w)
```

Da sich die Liste der Combo-Box in einer doch sehr eigenständigen WmShell befindet, gilt hier nicht mehr das übliche Schema zur Positionierung. Die Liste wird nicht automatisch mit der eigentlichen Combo-Box verschoben und gegebenenfalls zum Symbol verkleinert, sondern ist völlig autark. Außerdem sind die Koordinaten der WmShell nicht mehr wie gewohnt relativ zum jeweiligen Eltern-Widget, sondern statt dessen absolut (genau genommen dieses aber nur relativ zum root-Fenster). `DoDropDownLayout()` ermittelt daher die absolute Position des Combo-Box-Widgets und positioniert die WmShell derart, daß die Liste genau unterhalb der Eingabezeile erscheint.

```
XtGeometryResult QueryGeometry(XmComboBoxWidget w, XtWidgetGeometry *Request,
XtWidgetGeometry *Reply)
```

Auch hierbei handelt es sich um eine Methode, die vom X-Toolkit aufgerufen wird. Hierüber erfragt das Elternwidget, wie groß die Combo-Box sein möchte (Lieblingsgröße). Nicht umsonst meint die Xt-Dokumentation hierzu wörtlich: „*The parent may call this function when it want to know...*“. Die Betonung liegt dabei auf „*may*“, denn es bleibt im Ermessen des Elternwidgets, ob es die Xt-Funktion `XtQueryGeometry()` aufruft... oder eben nicht. Und insbesondere Form-Widgets kümmern sich nicht im Geringsten darum, welche Wünsche denn ihre Kinder haben könnten.

Fazit (nach einem tiefen Seufzer): Der Vollständigkeit halber muß diese Methode implementiert werden. Aber vielleicht sollte man ja einfach einmal einen `printf()`-Aufruf in diese Methode hineinschreiben, um so herauszufinden, ob den wirklich irgend jemand `XtQueryGeometry()` verwendet und daraufhin dann automatisch eine eMail an das MIT schicken...

```
XtGeometryResult GeometryManager(XmComboBoxWidget w, XtWidgetGeometry *Request,
XtWidgetGeometry *Reply)
```

Zur Methode `QueryGeometry()`, die dann aufgerufen wird, wenn das Eltern-Widget sich tatsächlich um die Wünsche seiner Kinder kümmert, bildet `GeometryManager()` das Pendant. Wenn eines der Kinder seine Größe verändern möchte, so geht das (zumindest theoretisch) nur mit Pappis Zustimmung. (Weniger wohlerzogene Kinder können sich auch kurzerhand mit `XtConfigureWidget()` verändern.) Da es nun ja wirklich nicht angehen kann, daß da so ein dahergelaufenes Widget unserer Combo-Box Wünsche vortragen möchte, sagt `QueryGeometry()` grundsätzlich „Nein!“. Man hat ja schließlich auch so seine Prinzipien... Aber Spaß beiseite, es wäre wirklich nicht sinnvoll bei einer Combo-Box, wenn sich die Kinder in ihrer Größe auf eigenen Wunsch hin verändern. Aber eigentlich sollte dieses auch überhaupt nicht vorkommen bei der Combo-Box.

```
void Resize(XmComboBoxWidget w)
```

Endlich einmal eine wirklich überschaubare Methode ist `Resize()`. Hierin werden lediglich Position und Größe der Eingabezeile sowie der Pfeil-Schaltfläche entsprechend der neuen Größe der Combo-Box angepaßt. In bewährter Manier übernimmt diese Aufgabe `DoLayout()`. Das war's dann auch schon. Bündig, nicht wahr?

```
void ShellCallback(Widget w, XtPointer cbw, XEvent *event, Boolean *ContDispatch)
```

Mit der Callback-Routine `ShellCallback()` steigen wir nun in die Mysterien der Motif-Programmierung und das Wunder des Tastaturfokus' ein. Bei der Liste handelt es sich um die fix und fertig vorhandene Motif-Liste, die lediglich eine eigene Verpackung erhielt, indem sie in eine WmShell gestopft wurde, bei der `XmNoverrideRedirect` auf `True` gesetzt ist. Die Besonderheit an der wmShell ist damit, daß sie vom Window-Manager ignoriert wird und machen kann, was sie will. Um nun zu verhindern, daß die ausgeklappte Liste einer Combo-Box weiterhin irgendwo auf dem Bildschirm herumschwebt, nachdem die Applikation soeben in ein Ikon verwandelt[1] wurde, muß die Combo-Box eben darauf achten.

Damit diese Benachrichtigungen abgehört werden können, sucht `Initialize()` zuerst nach der nächsten Shell, die ein Elternwidget der Combo-Box ist. Für diese Shell wird danach ein Callback installiert, der immer dann ausgelöst wird, wenn das dazugehörige Fenster vom Bildschirm verschwindet[2]. Dieses „Verschwinden“ tritt immer dann ein, wenn die Shell entweder in ein Ikon verwandelt wird (ich bringe den Begriff „ikonifizieren“ nicht über die Lippen, er ist einfach zu schräg) oder aber gelöscht wird.

Daneben ist `ShellCallback()` aber auch noch für solche Benachrichtigungen zuständig, die daraus resultieren, daß die Größe der Shell verändert wurden. Ein einfacher Aufruf von `DoDropDownLayout()` bringt die Sache danach wieder ins Lot.

[1] *Tja, hier geht das zu wie im Märchen. Da werden Dinge einfach verwandelt, da tauchen Ikonen auf (na, wo die wohl wieder geklaut wurden?), und und und...*

[2] *Hoppla! Jetzt verschwinden auch noch Dinge, das wird mir doch bald zu bunt.*

```
Boolean SetValues(XmComboBoxWidget current, XmComboBoxWidget req, XmComboBoxWidget new,
ArgList args, Cardinal *NumArgs)
```

Die Schnittstelle zwischen dem Programm und den parametrisierbaren Variablen („Ressourcen") einer jeden Widgetinstanz bedarf bei der Combo-Box der besonderen Beachtung. Die Combo-Box besitzt eine ganze Reihe von Ressourcen, die in Wirklichkeit nach außen hin nur vorgespiegelt sind. Tatsächlich sind die davon betroffenen Ressourcen Eigentum eines der diversen Kinder, mit der sich so eine Combo-Box herumplagen muß. Zu diesem Zweck besitzt die Combo-Box eine Liste aller Ressourcen seiner Kinder, auf die von außen mittels `XtSetValues` zugegriffen werden kann. Übrigens sind dabei einige Ressourcen nur lesbar, Schreibzugriffe werden ignoriert. Welche dieses im Einzelnen sind, zeigte bereits die Ressourcen-Übersicht weiter vorne in diesem Dokument. Jedoch kann damit nicht verhindert werden, daß ein bösartiger Programmierer nicht doch direkt auf die Ressourcen eines der Kinder zugreift.

```
void GetValuesAlmost(XmComboBoxWidget w, ArgList args, Cardinal *NumArgs)
```

Wiederum die Schnittstelle zwischen Applikation und Widget, diesmal aber zum Auslesen von Daten aus dem Widget. Auch hier gilt wieder das bereits zur Methode `SetValues` Gesagte.

```
void EditFocusCallback(Widget w, XtPointer pClientData, XmAnyCallbackStruct *info)
```

Diese Routine versucht, das unangenehme Verhalten des ausgeklappten Listenfeldes zu umgehen, sich jedesmal zurückzuziehen, wenn der Mauszeiger versehentlich außerhalb des Dialogfensters bewegt wird, in dem sich diese Combo-Box befindet. Der Trick besteht nun einfach darin, sich zu merken, ob vor dem hier eingetroffenen `FocusOut` bereits das Dialogfenster selbst einen `FocusOut` erhielt. In diesem Fall wird diese Nachricht einfach übergangen und das Listenfeld bleibt ausgeklappt (so es denn in diesem Moment ausgeklappt ist). Verliert die Combo-Box jedoch den Fokus, weil dieser an ein anderes Widget im gleichen Dialogfenster weitergegeben wurde, so muß selbstverständlich die Liste eingeklappt werden, damit sie nicht weiter stören kann.

Zu diesem Zwecke signalisiert der Callback `ShellCallback`, ob bereits kurz zuvor hier ein `FocusOut` stattfand. Ist dann auch noch die Ressource `XmNpersistentDropDown` auf `True` gesetzt, so wird dieses Ereignis vermerkt, damit diese Routine dann erkennen kann, daß die Liste nicht einzuklappen ist.

```
void CBoxManager(Widget w, XEvent *Event, String *params, Cardinal *num_params)
```

Hierbei handelt es sich um die zentrale Action-Prozedur, die alle neu hinzugekommenen Translationen des Eingabefeldes bearbeitet. Der Trick bei der Combo-Box liegt nämlich darin, daß alleine das Eingabefeld dieses Widgets den Tastaturfokus erhält – niemals jedoch die Liste. Damit der Benutzer aber doch in der Liste umherwandern kann, müssen zu diesem Zweck die benötigten Tasten und Translationen dem Eingabefeld hinzugefügt werden. Hier erfolgt dann die entsprechende Verarbeitung.

```
void ListSelectionCallback(Widget w, XtPointer pClientData, XmAnyCallbackStruct *info)
```

Wird ein Eintrag in der Listbox mit der Maus angeklickt, so sorgt diese Routine dafür, daß der betreffende Eintrag aus der Liste in das Eingabefeld transferiert und die hierfür registrierten Callbacks ausgeführt werden. Dabei stützt sich diese Routine auf `CBoxManager` ab, welche die Hauptarbeit zu leisten hat.

```
void EditVerifyCallback(Widget w, XtPointer pClientData, XmTextVerifyCallbackStruct *info)
```

Diese Routine spielt eine wichtige Rolle bei Combo-Boxen mit nicht editierbarem Eingabefeld. Denn bei diesen Exemplaren wird hierüber die Schnellsuche per Anfangsbuchstaben realisiert. Der Trick dazu ist ganz einfach: Sobald der Benutzer irgendetwas in das Eingabefeld tippen möchte, ruft das Eingabefeld diesen Callback auf und möchte bestätigt haben, daß das Eingetippte in die Eingabezeile auch eingefügt werden darf. Bei der Schnellsuche wird dieses grundsätzlich verneint, statt dessen aber nach einem Eintrag mit dem passenden Anfangsbuchstaben gesucht und dieser dann als angewählt markiert.

Warum aber nun dieser Weg und nicht einfach die Translation „<Key>" überschrieben? Nun – im letzteren Falle würden *alle* Tasteneingaben umgeleitet, auch diejenigen, die das Eingabefeld selbst handhaben soll, wie beispielsweise „Cursor rechts“ und so weiter. Schuld an dieser Misere ist der Translationen-Manager, der dieser nachträglich registrierten „Joker“-Translation den Vorzug gegenüber allen bislang für dieses Widget bekannten Translationen gibt[3].

```
void EditChangedCallback(Widget w, XtPointer pClientDate, XmAnyCallbackStruct *info)
```

Während die vorangegangene Routine nur bei Combo-Boxen mit nicht veränderbarem Eingabefeldern aktiv wird, bekommt diese Routine erst Arbeit bei der Spezies Combo-Box mit veränderbarem Eingabefeld. Nach jeder Veränderung des Textes in der Eingabezeile überprüft diese Prozedur, ob der Text zufällig mit einem der Einträge im Listenfeld übereinstimmt. Ist dieses der Fall, so wird der entsprechende Eintrag in der Liste hervorgehoben.

```
void Initialize(Widget request, XmComboBoxWidget new, ArgList wargs, Cardinal *ArgCount)
```

Hier muß nun die Hauptarbeit geleistet werden, sobald jemand in seinem Programm eine Combo-Box einrichtet. Dabei müssen die benötigten Kinder-Widgets erzeugt werden, als da wären: ein XmTextField – die Eingabezeile. Daneben ein XmArrowButton, der es dem Benutzer ermöglicht, mit der Maus die Liste auszuklappen. Hinzu gesellt sich weiterhin eine wmShell, die ihrerseits ein XmScrolledWindow aufnimmt, in das hinein dann die XmList kommt.

Des weiteren erfolgt hier schon einmal die Layout-Anordnung mit den bislang vorgegebenen Geometriedaten. Dieses ist erforderlich, da beispielsweise das XmForm-Widget nicht im Traum daran denkt, vielleicht einmal ein XtQueryGeometry an seine Kinder abzusetzen. Somit wird verhindert, daß das Eingabefeld und die Pfeilschaltfläche noch nicht an ihrem richtigen Platz innerhalb der Combo-Box stehen, sobald das Widget auf dem Bildschirm erscheint.

```
void InitMirrorRessources(XmComboBoxWidget w)
```

Diese Routine ist eines der Sahnestückchen dieses Widgets. Sie sorgt dafür, daß während der Initialisierung der Combo-Box solche gespiegelten Ressourcen, für die Einträge in der Ressourcen-Datenbank vorhanden sind, mit den neuen Einstellungen versorgt werden.

Dabei wird anhand einer Liste mit den entsprechenden Ressourcen-Namen die Datenbank abgefragt und im Erfolgsfall die Daten aus dem reinen textuellen Format der Datenbank in das benötigte Zielformat konvertiert. Als netter Trick sorgt dann ein `XtVaSetValues` dafür, daß die Daten an die Kinder geleitet werden. Dabei wird als Zielwidget die Combo-Box selbst angegeben, denn auf diese Weise kann `SetValues` die Arbeit übernehmen, die Ressourcen richtig auf die Kinder zu verteilen und die erforderliche Logik muß nicht noch einmal dupliziert werden.

[3] *Das war im Übrigen eines dieser „schönen“ Probleme, über denen ich teilweise ganz schön lange brüten mußte, um ihre Ursache zu erkennen und dann eine Lösung auszuhecken.*