der Window-Manager KDE hingegen ein X11-Client. Das X11-System zeichnet aus, dass der Client auf einem anderen Rechner laufen kann als der Server, sofern beide über ein Netzwerk wie beispielsweise TCP/IP[3] verbunden sind. So ist es z.B. möglich, dass ein Linux-PC über das Internet mit einer Unix-Workstation verbunden ist und auf der Workstation ein Grafikprogramm läuft, das an dem PC bedient wird.
Unter Linux kommt die freie X11-Variante XFree86 zum Einsatz, die ursprünglich für 80x86 Prozessoren entwickelt wurde und als offener Quelltext verfügbar ist. Die in dem Paket enthaltene Funktionsbibliothek `libX11` bietet eine Vielzahl von Funktionen zur Manipulation von Grafikobjekten. Dennoch eignet sich diese Bibliothek allein nicht zum Erstellen von Programmen mit grafischer Benutzerschnittstelle. Es gibt eine Reihe weiterer Bibliotheken, die Funktionen für komplexe Bedienelemente wie beispielsweise Menüs beinhalten. Einige dieser Bibliotheken – auch als *Toolkits* bezeichnet – sind:

- Tk
- GTK+ (The Gimp Toolkit)
- XView
- Qt
- Motif (bzw. LessTif, ein Open Source Motif-Ersatz)
- Athena Widget

Das „Look & Feel“ dieser Bibliotheken ist sehr unterschiedlich, was sich auf das Erscheinungsbild der verschiedenen Programme unter Linux auswirkt.

8.2 XView und OpenLook

XView (Abk. für **X** Window System based **V**isual **I**ntegrated **E**nvironment for **W**orkstations) wurde von der Firma *Sun Microsystems* als Nachfolger des *SunView Toolkit*s entwickelt. Die Programmierung ist objektorientiert und auch für Einsteiger leicht zu erlernen – Kenntnisse in C++ werden dazu nicht benötigt. Das Erscheinungsbild und die Funktionalität von Programmen auf Basis der XView-Bibliothek entsprechen dem *OpenLook*-Standard, der von den Firmen Sun Microsystems und

[3] Transfer Control Protocol / Internet Protocol

AT&T als Standard-Benutzerinterface für Unix *System V* entwickelt wurde. Die Window-Manager „`olwm`“ und „`olvwm`“ basieren auf XView und zeigen sich dementsprechend im OpenLook-Stil (siehe Bild 8.1).

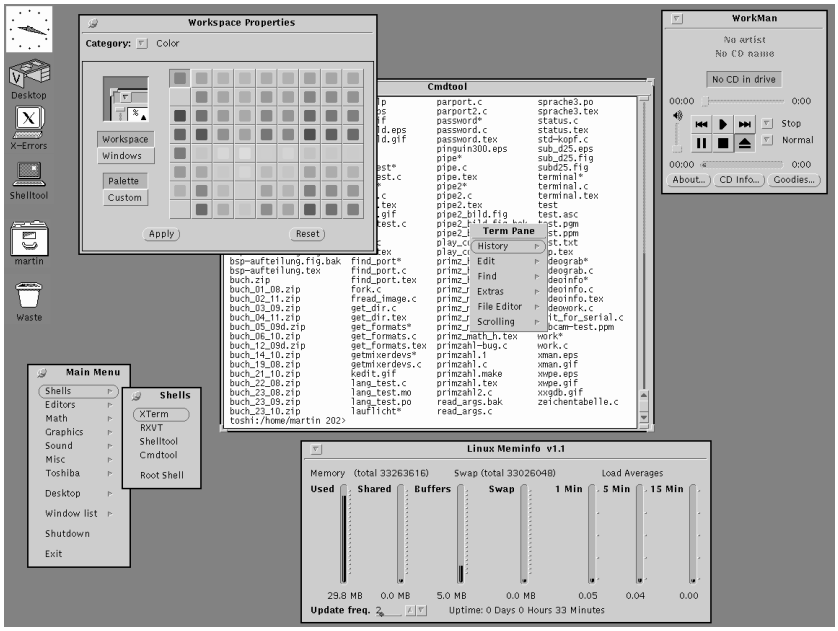

Bild 8.1: Der Window-Manager „`olvwm`“ und einige XView-Programme

XView-Programme kennen eine ganze Reihe von Kommandozeilenoptionen, die von der `libxview` automatisch ausgewertet werden. Mit der Option „`-Wi`“ wird ein Programm beispielsweise als *Icon* gestartet. Infos zur `libxview` und den Kommandozeilenoptionen sind in der entsprechenden „`man`“-Seite:[4]

```
man -l /usr/openwin/man/man1/xview.*
```

8.2.1 Ein Fenster öffnen

Als Einstieg in die Programmierung mit der XView-Bibliothek soll hier zunächst mit einem kleinen Programm das Öffnen eines Fensters demon-

[4] Im Prinzip sollte „`man xview`“ genügen; wenn jedoch das Programm „`xli`“ installiert ist, so wird damit unter Umständen dessen Beschreibung angezeigt.

striert werden:

```
 1  /*
 2      frame.c - Ein einfaches XView-Fenster oeffnen
 3  */
 4
 5  # include <stdio.h>
 6  # include <xview/frame.h>
 7
 8  int main(int argc, char *argv[])
 9   {
10    Frame frame;
11
12    xv_init(XV_INIT_ARGC_PTR_ARGV, &argc, argv, 0);
13    frame = xv_create(0, FRAME, 0);
14    xv_main_loop(frame);
15    return(0);
16   }
```

Zum Übersetzen dieses Programms müssen die drei Funktionsbibliotheken `libX11`, `libxview` und `libolgx` eingebunden werden. Ferner müssen die Pfade für die Include-Dateien und die Bibliotheken angegeben werden:

```
gcc -I/usr/openwin/include -L/usr/X11/lib
-L/usr/openwin/lib -lX11 -lxview -lolgx frame.c
-o frame
```

Man kann sich hier eine Menge Tipparbeit sparen, indem man das Programm `make` verwendet (siehe Abschnitt 1.2.3) oder ein Shell-Skript mit der Compile-Anweisung erstellt. Ein solches Skript steht als Download zum Buch unter „tools“ zur Verfügung (siehe Anhang A1). Nach dem Übersetzen des Quelltextes kann das Programm mit

```
frame
```

gestartet werden. Es erscheint ein leeres Fenster mit dem Titel „No name“ (auf ein Bild, das dieses leere Fenster zeigt, sei hier verzichtet). Wird das Fenster mit der entsprechenden Schaltfläche in dem Titelbalken geschlossen, wird auch das Programm automatisch beendet.
Auch wenn dieses Programm noch keine sinnvolle Funktion hat, lässt sich daran das Konzept von XView erläutern. In Zeile 10 wird eine Variable

mit dem Namen `frame` vom Typ `Frame` definiert. Was man im allgemeinen Sprachgebrauch als Fenster bezeichnet – nämlich ein rechteckiges Objekt mit einem Titelbalken und den „Dekorationen“ des Window-Managers –, wird unter XView als „Frame“ (Rahmen) bezeichnet. Den Begriff „Window“ gibt es in XView ebenfalls, er hat jedoch eine andere Bedeutung.
Mit dem Aufruf der Funktion `xv_init()` in Zeile 12 werden die Kommandozeilenparameter ausgewertet. Dabei werden alle XView-Optionen wie z.B. „`-Wi`“ aus dem Feld `argv[]` entfernt und der Wert von `argc` entsprechend reduziert, sodass anschließend nur Parameter und Optionen übrig bleiben, die XView unbekannt sind. Ist das Modifizieren der Variablen `argc` und `argv` nicht gewünscht, muss die Funktion `xv_init()` wie folgt aufgerufen werden:

```
xv_init(XV_INIT_ARGS, argc, argv, 0L);
```

Anmerkung: Die Funktion `xv_init()` liefert als Rückgabewert ein Objekt vom Typ `Xv_Server`, das Informationen über den kontaktierten X-Server enthält. Kann kein X-Server kontaktiert werden, wird das Programm beim Aufruf von `xv_init()` automatisch mit einer Fehlermeldung beendet.
In Zeile 13 wird das Fenster erzeugt, aber noch nicht geöffnet. Hier erkennt man die objektorientierte Struktur der XView-Bibliothek, da es sich bei dem Typ `Frame` um ein XView-Objekt (Typ `Xv_opaque`) handelt – ebenso wie bei Schaltflächen oder Menüs. Diese Objekte sind „undurchsichtig“ (*opaque*), es kann nur mit den XView-Funktionen darauf zugegriffen werden. Alle XView-Objekte werden mit `xv_create()` erzeugt, wobei der erste Parameter ebenfalls ein XView-Objekt ist, das als „Besitzer“ des neuen Objektes eingetragen wird. Diese Verkettung bewirkt, dass durch das Schließen des Besitzers auch die untergeordneten Objekte geschlossen werden. Eine 0 als erster Parameter bewirkt, dass das neue Objekt keinem anderen untergeordnet ist.
Mit dem zweiten Parameter wird der Funktion `xv_create()` der Typ des zu erzeugenden Objektes angegeben – bei dem Beispielprogramm ist das ein `FRAME`. Danach folgt eine mit 0 abgeschlossene Liste von Attributen für das neue Objekt. Ändert man die Zeile 13 des Programms beispielsweise wie folgt ab

```
frame = xv_create(0, FRAME, FRAME_LABEL, "Hallo", 0);
```

und übersetzt das Programm neu und startet es, so erhält das Fenster jetzt den Titel „Hallo“. Eine Liste der möglichen Attribute für ein

Fenster findet sich auf Seite 192. Kann die Funktion `xv_create()` das angegebene Objekt nicht erzeugen, liefert sie den Rückgabewert 0. Korrekterweise sollte also der Rückgabewert vor der weiteren Verwendung überprüft werden.
Wie bereits erwähnt, wird das Fenster durch `xv_create()` zwar erzeugt, aber noch nicht geöffnet; es bleibt also zunächst unsichtbar. Das liegt an dem Client-Server-Konzept von X11. Erst wenn ein Prozess auch Ereignisse – z.B. das Anklicken mit der Maus – entgegennimmt, werden auch die Fenster dargestellt. Dies geschieht bei XView mit der Funktion `xv_main_loop()` in Zeile 14. Diese Funktion richtet den so genannten *Notifier* ein, der X11-Ereignisse annimmt und entsprechende Funktionen aufruft. Die Funktion `xv_main_loop()` kehrt erst zurück, wenn das als Parameter angegebene Objekt geschlossen wird.

Die Attribute des Frame-Objektes

Wie bereits oben beschrieben, kann einem XView-Objekt bei der Erzeugung mittels `xv_create()` eine Liste von Attributen mitgegeben werden. Tabelle 8.1 zeigt eine Auswahl der Attribute für das Objekt „`Frame`“. Einige dieser Attribute zeigen nicht bei jedem Window-Manager eine Auswirkung.

Tabelle 8.1: Die Attribute für das XView-Objekt „`Frame`“

Attribut	Parameter	Beschreibung
`FRAME_BUSY`	`TRUE` oder `FALSE`	Uhr als Mauszeiger
`FRAME_CLOSED`	`TRUE` oder `FALSE`	Fenster als Icon
`FRAME_HEADER`	Zeichenkette	wie `FRAME_LABEL`
`FRAME_ICON`	Icon-Bitmap	Grafik für Icon
`FRAME_LABEL`	Zeichenkette	Titel
`FRAME_LEFT_FOOTER`	Zeichenkette	linke Fußzeile
`FRAME_NO_CONFIRM`	`TRUE` oder `FALSE`	beim Schließen fragen
`FRAME_RIGHT_FOOTER`	Zeichenkette	rechte Fußzeile
`FRAME_SHOW_FOOTER`	`TRUE` oder `FALSE`	Fußzeile zeigen
`FRAME_SHOW_RESIZE_CORNER`	`TRUE` oder `FALSE`	Größe variabel
`FRAME_SHOW_LABEL`	`TRUE` oder `FALSE`	Titelbalken anzeigen
`FRAME_MIN_SIZE`	Breite, Höhe	min. Größe
`FRAME_MAX_SIZE`	Breite, Höhe	max. Größe

Neben den speziellen Attributen für Frames gibt es allgemeine Attribute, die sich auf verschiedene XView-Objekte anwenden lassen. Tabelle 8.2 zeigt eine Auswahl dieser Attribute.

Tabelle 8.2: Attribute für verschiedene XView-Objekte

Attribut	Parameter	Beschreibung
`XV_X`	X-Position	X-Position vorgeben
`XV_Y`	Y-Position	Y-Position vorgeben
`XV_SHOW`	`TRUE` oder `FALSE`	Objekt darstellen
`XV_WIDTH`	Breite	Breite des Objektes vorgeben
`XV_HEIGHT`	Höhe	Höhe des Objektes vorgeben

Beispiel: In dem obigen Programmbeispiel soll das Fenster an der Position $(X, Y) = (20, 10)$ mit einer Breite von 300 und einer Höhe von 200 Punkten geöffnet werden. Ferner soll es den Titel „Mein Fenster“ und eine Fußzeile mit dem Inhalt „links“ und „rechts“ erhalten. Dazu muss die Zeile 13 des Quelltextes wie folgt erweitert werden:

```
frame = xv_create(0, FRAME,
  XV_X,                20,
  XV_Y,                10,
  XV_WIDTH,            300,
  XV_HEIGHT,           200,
  FRAME_LABEL,         "Mein Fenster",
  FRAME_LEFT_FOOTER,   "links",
  FRAME_RIGHT_FOOTER, "rechts",
  FRAME_SHOW_FOOTER,   TRUE,
  0);
```

Die Reihenfolge der Attribute ist dabei beliebig, die Liste muss aber unbedingt mit 0 abgeschlossen werden. Bild 8.2 zeigt das Resultat.

8.2.2 Attribute abfragen und ändern

Die Attribute eines XView-Objekts können nicht nur bei Erzeugung des Objektes festgelegt, sondern auch nachträglich geändert werden. Dies geschieht mit der Funktion `xv_set()`:

```
xv_set(xv_object, attribute, setting, ...);
```

Es können mit einem `xv_set()`-Aufruf beliebig viele Attribute modifiziert werden; die Liste muss wie bei `xv_create()` mit einer 0 abgeschlossen werden, Beispiel:

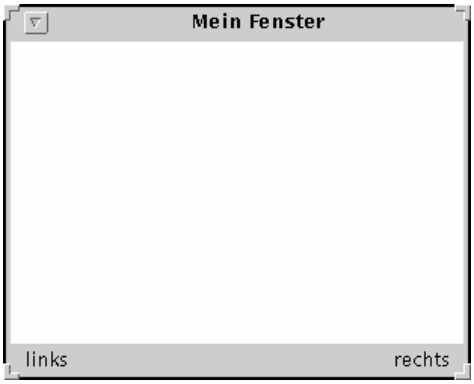

Bild 8.2: Ein „Frame“ mit verschiedenen Attributen

```
xv_set(frame,
  FRAME_LABEL, "neuer Titel",
  XV_WIDTH,    400,
  0);
```

Da XView-Objekte „undurchsichtig“ sind, also nicht wie eine Struktur ausgelesen werden können, muss es eine Funktion zum Abfragen eines Attributes des Objektes geben. Das ist Aufgabe der Funktion `xv_get()`:

```
setting = xv_get(xv_object, attribute);
```

Beispiel:

```
titel = (char *)xv_get(frame, FRAME_LABEL);
```

8.2.3 Bedienfelder einrichten

Das Beispielprogramm aus Abschnitt 8.2.1 macht noch recht wenig Sinn, da das erzeugte Fenster weder Ein- noch Ausgabe-Elemente enthält. Unter XView werden solche Elemente wie z.B. Schaltflächen nicht direkt auf dem Fenster (Frame) platziert, sondern erfordern ein Bedienfeld. Das zugehörige XView-Objekt heißt *Panel* und wird ebenso wie ein Frame mit `xv_create()` erzeugt:

```
Frame frame;
Panel panel;

panel = xv_create(frame, PANEL, 0);
```

Da das Panel Teil eines Frames sein muss, wird bei dem `xv_create()`-Aufruf der Frame als „Besitzer“ des Panels angegeben. Um die Funktion des Bedienfeldes zu demonstrieren, sei das folgende Programm betrachtet, das auf dem Bedienfeld zwei *Message*-Objekte erzeugt:

```
 1  /*
 2      panel.c - Fenster mit Bedienfeld
 3  */
 4
 5  # include <stdio.h>
 6  # include <xview/frame.h>
 7  # include <xview/panel.h>
 8
 9  int main(int argc, char *argv[])
10   {
11    Frame frame;
12    Panel panel;
13    Panel_message_item message1, message2;
14
15    xv_init(XV_INIT_ARGC_PTR_ARGV, &argc, argv, 0);
16
17    frame = xv_create(0, FRAME,
18      XV_WIDTH,  200,
19      XV_HEIGHT, 100,
20      0);
21
22    panel = xv_create(frame, PANEL, 0);
23
24    message1 = xv_create(panel, PANEL_MESSAGE,
25      PANEL_LABEL_STRING, "Hier eine Nachricht.",
26      0);
27
28    message2 = xv_create(panel, PANEL_MESSAGE,
29      PANEL_LABEL_STRING, "Und hier eine 2. Nachricht",
30      0);
31
32    xv_main_loop(frame);
33    return(0);
34   }
```

Nachdem das Programm übersetzt und gestartet wurde, erscheint das in Abbildung 8.3 dargestellte Fenster. Im Gegensatz zu Bild 8.2 ist das

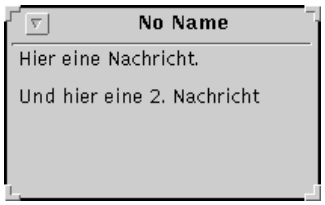

Bild 8.3: Ein Bedienfeld (Panel) mit zwei Message-Objekten

Fenster jetzt nicht mehr leer, sondern grau gefüllt und enthält zwei Textzeilen. Die beiden Message-Objekte sollen hier lediglich die Funktion des Panel-Objekts verdeutlichen – eine Beschreibung des Message-Objekts selbst folgt in Abschnitt 8.2.7.

Attribute des Panel-Objekts

Das XView-Objekt „Panel“ besitzt unter anderem die folgenden Attribute:

Attribut	Parameter	Beschreibung
`PANEL_LAYOUT`	`PANEL_VERTICAL` oder `PANEL_HORIZONTAL`	Anordnung der Bedienelemente
`PANEL_BORDER`	`TRUE` oder `FALSE`	Bedienfeld einrahmen
`PANEL_BLINK_CARET`	`TRUE` oder `FALSE`	Eingabe-Cursor blinkt
`PANEL_CARET_ITEM`	XView-Objekt	aktives Text-Eingabefeld
`PANEL_ITEM_X_GAP`	X-Abstand (Pixel)	horizontaler Abstand der Bedienelemente
`PANEL_ITEM_Y_GAP`	Y-Abstand (Pixel)	vertikaler Abstand der Bedienelemente

8.2.4 Schaltflächen (Buttons) einrichten

Im nächsten Schritt soll nun eine Schaltfläche eingerichtet werden, mit der das Programm beendet werden kann. Eine Schaltfläche wird in XView mit dem Objekt „Button“ erzeugt. Dazu sei das folgende Programm betrachtet:

```
1  /*
2       quit.c - Fenster mit einer Schaltflaeche
3                zum Beenden des Programms
```

```
 4  */
 5
 6  # include <stdio.h>
 7  # include <xview/frame.h>
 8  # include <xview/panel.h>
 9
10  void quit_proc(Panel_item item, Event *event)
11   {
12    Frame frame;
13
14    frame = xv_get(xv_get(item, XV_OWNER), XV_OWNER);
15    xv_destroy_safe(frame);
16    return;
17   }
18
19  int main(int argc, char *argv[])
20   {
21    Frame frame;
22    Panel panel;
23    Panel_button_item quit_button;
24
25    xv_init(XV_INIT_ARGC_PTR_ARGV, &argc, argv, 0);
26
27    frame = xv_create(0, FRAME,
28      XV_WIDTH,  200,
29      XV_HEIGHT, 100,
30      0);
31
32    panel = xv_create(frame, PANEL, 0);
33
34    quit_button = xv_create(panel, PANEL_BUTTON,
35      PANEL_ITEM_X,       20,
36      PANEL_ITEM_Y,       50,
37      PANEL_LABEL_STRING, "Programm beenden",
38      PANEL_NOTIFY_PROC,  quit_proc,
39      0);
40
41    xv_main_loop(frame);
42    return(0);
43   }
```

Dieses Programm erzeugt das in Bild 8.4 dargestellte Fenster. Wird die Schaltfläche mit der Maus angeklickt, so wird das Fenster geschlossen und das Programm beendet. Betrachten wir zunächst das Hauptpro-

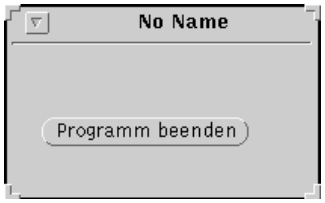

Bild 8.4: Beispiel für eine Schaltfläche (Button)

gramm (Zeile 19 bis 43): Nach dem Erzeugen des Fensters und des Bedienfeldes wird in den Zeilen 34 bis 39 ein Button-Objekt erzeugt. Dabei wird als „Besitzer“ das Bedienfeld (`panel`) angegeben. Die Position des Buttons wird mit den Attributen `PANEL_ITEM_X` und `PANEL_ITEM_Y` explizit vorgegeben[5] und mit `PANEL_LABEL_STRING` wird der Text der Schaltfläche eingestellt. In Zeile 38 erfolgt dann die Angabe der so genannten *Notify Procedure*, der Funktion, die bei Betätigung der Schaltfläche ausgeführt werden soll. In diesem Fall ist das die Funktion `quit()`, die in den Zeilen 10 bis 17 definiert wird.
Nach Erzeugung der Schaltfläche wird wie in den Beispielen zuvor die Funktion `xv_main_loop()` aufgerufen. Diese wartet auf Ereignisse, die das als Parameter angegebene Objekt (oder eines der daran angebundenen Objekte) betreffen und führt dann die entsprechende Notify-Prozedur aus. Dieser Mechanismus wird als *Callback* bezeichnet, die Notify-Prozeduren werden auch *Callback-Funktionen* genannt.
Betrachten wir nun die Funktion `quit()` ab Zeile 10. Wird diese von `xv_main_loop()` aufgerufen, so werden zwei Parameter übergeben. Der erste Parameter enthält das Objekt, das betätigt wurde. Das ist hilfreich, wenn die gleiche Funktion für verschiedene Objekte verwendet wird. Der zweite Parameter enthält einen Zeiger auf das Ereignis (*Event*), das zum Aufruf der Funktion führte. Hierüber lässt sich beispielsweise die Mausposition bei Betätigung der Schaltfläche feststellen.
In Zeile 14 wird mit Hilfe von zwei verschachtelten `xv_get()`-Aufrufen das Frame-Objekt ermittelt, denn der Frame ist „Besitzer“ (`XV_OWNER`) des Panels, welches wiederum Besitzer des Buttons ist. (Im nächsten

[5] Diese Positionsangaben sind relativ zur linken, oberen Ecke des Panels.

Abschnitt wird eine elegantere Methode zur Bestimmung des zugehörigen Frame-Objekts vorgestellt.) Mit der Anweisung `xv_destroy_safe()` wird das Frame-Objekt „zerstört“. Dadurch wird das Fenster geschlossen, was wiederum die Funktion `xv_main_loop()` im Hauptprogramm beendet.
Wird ein Objekt mit `xv_destroy_safe()` oder `xv_destroy()`[6] zerstört, werden auch die damit verknüpften Objekte zerstört – bei dem obigen Beispiel also das Bedienfeld und die Schaltfläche.

Attribute des Button-Objekts

Eine Auswahl der möglichen Attribute für das XView-Objekt „Button“ ist in der Tabelle 8.3 aufgelistet.

Tabelle 8.3: Attribute für Schaltflächen (Buttons)

Attribut	Parameter	Beschreibung
`PANEL_ITEM_X`	X-Position (Pixel)	Position innerhalb des Bedienfeldes
`PANEL_ITEM_Y`	Y-Position (Pixel)	Position innerhalb des Bedienfeldes
`PANEL_LABEL_STRING`	Zeichenkette	Beschriftung der Schaltfläche
`PANEL_NOTIFY_PROC`	Funktion	Aktion bei Betätigung
`PANEL_ITEM_COLOR`	Farbindex	Farbe der Schaltfläche
`PANEL_ITEM_DEAF`	`TRUE` oder `FALSE`	Button reagiert nicht
`PANEL_INACTIVE`	`TRUE` oder `FALSE`	Button nicht bedienbar
`PANEL_BUSY`	`TRUE` oder `FALSE`	Button ist gerade aktiv
`PANEL_NEXT_COL`	Abstand (Pixel)	neue Spalte beginnen mit vorgegebenem Abstand (-1 für `PANEL_ITEM_X_GAP`)
`PANEL_NEXT_ROW`	Abstand (Pixel)	neue Zeile beginnen mit vorgegebenem Abstand (-1 für `PANEL_ITEM_Y_GAP`)

Das folgende Bild zeigt die Auswirkung einiger dieser Attribute. Die

[6] Der Unterschied zwischen `xv_destroy()` und `xv_destroy_safe()` besteht darin, dass XView bei Letzterem das „saubere“ Schließen des Objektes sicherstellt. Daher sollte in der Regel `xv_destroy_safe()` verwendet werden.

Schaltfläche unten rechts ist „inaktiv“ geschaltet:

Bevorzugte Schaltflächen

Häufig stehen in einem Fenster mehrere Schaltflächen zur Auswahl, von denen eine jedoch *bevorzugt* betätigt wird. Das bedeutet, dass das Drücken der Taste Return automatisch diese Schaltfläche auslöst. Bei der Sicherheitsabfrage beim Beenden eines Programms könnte beispielsweise die Schaltfläche „Nein“ voreingestellt sein. Bevorzugte Schaltflächen werden durch eine zusätzliche Umrandung gekennzeichnet.
Zum Einstellen einer bevorzugten Schaltfläche sind zwei Schritte notwendig. Bei der Schaltfläche selbst muss das Attribut `PANEL_ACCEPT_KEYSTROKE` eingestellt sein, damit auch Tastatureingaben von dieser Schaltfläche registriert werden. Darüber hinaus muss bei dem zugehörigen Bedienfeld die Schaltfläche als `PANEL_DEFAULT_ITEM` eingerichtet sein. Mit diesen Erweiterungen sieht das Beispielprogramm von Seite 196 dann wie folgt aus:

```
1  /*
2      quit2.c - Fenster mit einer Schaltflaeche
3                zum Beenden des Programms
4  */
5
6  # include <stdio.h>
7  # include <xview/frame.h>
8  # include <xview/panel.h>
9
10 void quit_proc(Panel_item item, Event *event)
11  {
12   Frame frame;
13
14   frame = xv_get(xv_get(item, XV_OWNER), XV_OWNER);
15   xv_destroy_safe(frame);
16   return;
17  }
```

```
18
19  int main(int argc, char *argv[])
20   {
21    Frame frame;
22    Panel panel;
23    Panel_button_item quit_button;
24
25    xv_init(XV_INIT_ARGC_PTR_ARGV, &argc, argv, 0);
26
27    frame = xv_create(0, FRAME,
28      XV_WIDTH,  200,
29      XV_HEIGHT, 100,
30      0);
31
32    panel = xv_create(frame, PANEL, 0);
33
34    quit_button = xv_create(panel, PANEL_BUTTON,
35      PANEL_ITEM_X,           20,
36      PANEL_ITEM_Y,           50,
37      PANEL_LABEL_STRING,     "Programm beenden",
38      PANEL_NOTIFY_PROC,      quit_proc,
39      PANEL_ACCEPT_KEYSTROKE, TRUE,
40      0);
41
42    xv_set(panel, PANEL_DEFAULT_ITEM, quit_button, 0);
43
44    xv_main_loop(frame);
45    return(0);
46   }
```

Ergänzt wurden hier die Zeilen 39 und 42. Wenn Sie das Programm übersetzen und ausführen, kann es durch Drücken der Return-Taste beendet werden.

8.2.5 Daten an eine Callback-Funktion weiterleiten

In dem Beispielprogramm aus dem vorherigen Abschnitt benötigte die Callback-Funktion für die Schaltfläche „Programm beenden“ das Frame-Objekt. Über zwei verschachtelte `xv_get()`-Aufrufe wurde das Frame-Objekt von dem Button-Objekt abgeleitet. Nicht immer lassen sich jedoch Daten, die eine Callback-Funktion benötigt, aus den Parametern

ermitteln. Man könnte solche Daten als *globale* Variablen ablegen und auf diese Weise für alle Funktionen verfügbar machen. Die Verwendung globaler Variablen kennzeichnet aber keinen guten Programmierstil und sollte auf ein Minimum reduziert werden.
XView bietet zur Lösung dieses Problems zwei Möglichkeiten:

1. Wird nur *eine* Variable innerhalb der Callback-Funktion benötigt, so kann diese Variable an das Objekt angebunden werden, das die Callback-Funktion auslöst. Dies geschieht mit dem Attribut `PANEL_CLIENT_DATA`. Mit „`xv_get(`*Objekt*`, PANEL_CLIENT_DATA)`“ kann die Variable dann innerhalb der Callback-Funktion ausgewertet werden.

2. Muss man *mehrere* Variablen für die Callback-Funktion verfügbar machen, können diese jeweils mit Hilfe des Attributs `XV_KEY_DATA` an das Objekt geknüpft werden, wobei dieses Attribut neben der zu übergebenden Variable noch die Angabe eines *Keys* (Schlüssels) erfordert. Als Key dient dabei eine beliebige, konstante Zahl. Mit unterschiedlichen Keys lassen sich somit beliebig viele Daten an ein Objekt anhängen.

Das folgende Programm verwendet beide Möglichkeiten, die erste Schaltfläche benutzt `PANEL_CLIENT_DATA`, die zweite `XV_KEY_DATA`. Man beachte die Zeilen 9 und 10, in denen die „Keys“ festgelegt werden.

```
 1  /*
 2      key_data.c - Daten an Callback-Funktion uebergeben
 3  */
 4
 5  # include <stdio.h>
 6  # include <xview/frame.h>
 7  # include <xview/panel.h>
 8
 9  # define KEY_FRAME 1
10  # define KEY_ARGV 2
11
12  void button1_proc(Panel_item button, Event *event)
13   {
14    Frame frame;
15    char **argv;
16
17    frame = xv_get(xv_get(button, XV_OWNER), XV_OWNER);
```

```
18    argv = (char **)xv_get(button, PANEL_CLIENT_DATA);
19    xv_set(frame,
20      FRAME_LEFT_FOOTER,  "mit 'CLIENT_DATA':",
21      FRAME_RIGHT_FOOTER, argv[0],
22      FRAME_SHOW_FOOTER,  TRUE,
23      0);
24    return;
25   }
26
27  void button2_proc(Panel_item button, Event *event)
28   {
29    Frame frame;
30    char **argv;
31
32    frame = xv_get(button, XV_KEY_DATA, KEY_FRAME);
33    argv = (char **)xv_get(button,
34                           XV_KEY_DATA, KEY_ARGV);
35    xv_set(frame,
36      FRAME_LEFT_FOOTER,  "mit 'KEY_DATA':",
37      FRAME_RIGHT_FOOTER, argv[0],
38      FRAME_SHOW_FOOTER,  TRUE,
39      0);
40    return;
41   }
42
43  int main(int argc, char *argv[])
44   {
45    Frame frame;
46    Panel panel;
47
48    xv_init(XV_INIT_ARGC_PTR_ARGV, &argc, argv, 0);
49
50    frame = xv_create(0, FRAME, 0);
51
52    panel = xv_create(frame, PANEL, 0);
53
54    xv_create(panel, PANEL_BUTTON,
55      PANEL_LABEL_STRING, "mit `CLIENT_DATA`",
56      PANEL_NOTIFY_PROC,  button1_proc,
57      PANEL_CLIENT_DATA,  argv,
58      0);
```

```
59
60    xv_create(panel, PANEL_BUTTON,
61      PANEL_LABEL_STRING, "mit ‘KEY_DATA‘",
62      PANEL_NOTIFY_PROC,  button2_proc,
63      XV_KEY_DATA,        KEY_FRAME, frame,
64      XV_KEY_DATA,        KEY_ARGV, argv,
65      0);
66
67    xv_main_loop(frame);
68    return(0);
69   }
```

8.2.6 Anpassen der Fenster- und Bedienfeldgröße

Wird ein Fenster oder ein Bedienfeld mit `xv_create()` erzeugt, hat dieses entweder die mit `XV_WIDTH` und `XV_HEIGHT` vorgegebene oder – falls diese Attribute nicht gesetzt werden – eine voreingestellte Größe. Da der Anwender später die Möglichkeit hat, die Schriftgröße (und damit auch die Größe von Schaltflächen) in einem Fenster mit Hilfe der Kommandozeilenoption „`-scale large`“ zu verändern, sollten Abmessungen von Fenstern und Bedienfeldern möglichst nicht fest vorgegeben werden. Stattdessen bietet sich die Nutzung spezieller XView-Funktionen an, mit denen sich die Fenster- und Bedienfeldgröße optimal anpassen lässt:

```
window_fit(Xv_opaque object);
window_fit_width(Xv_opaque object);
window_fit_height(Xv_opaque object);
```

Als XView-Objekt kann bei allen drei Funktionen ein Frame oder ein Panel angegeben werden. In der Regel werden zunächst alle Bedienelemente eines Pannels erzeugt und anschließend die Größe des Panels mit `window_fit()` so eingestellt, dass gerade eben alle Bedienelemente darin Platz haben. Häufig erfolgt danach ein Aufruf von `window_fit()` für das Fenster, in dem sich das Bedienfeld befindet, sodass auch dieses an die Bedienfeldgröße angepasst wird. Beispiel:

```
Frame frame;
Panel panel;

xv_init(XV_INIT_ARGC_PTR_ARGV, &argc, argv, 0);
frame = xv_create(0, FRAME, 0);
panel = xv_create(frame, PANEL, 0);
```

```
xv_create(panel, PANEL_BUTTON,
  PANEL_LABEL_STRING, "Test",
  PANEL_NOTIFY_PROC,  my_proc,
  0);
window_fit(panel);
window_fit(frame);
xv_main_loop(frame);
```

8.2.7 Das Message-Objekt

Eines der möglichen Elemente auf einem Bedienfeld ist das Message-Objekt. Es kann zur Ausgabe von Status-Meldungen oder als (zusätzliche) Beschriftung anderer Elemente des Bedienfeldes verwendet werden. Das Message-Objekt kann nur Text darstellen und hat keine Eingabefunktion. Das Beispielprogramm aus Abschnitt 8.2.3 zeigt die Verwendung dieses Objektes. Tabelle 8.4 zeigt mögliche Attribute für ein Message-Objekt.

8.2.8 Auswahlfelder

Neben den bereits beschriebenen Schaltflächen (Buttons) gibt es noch eine ganze Reihe weiterer Eingabe-Elemente, die innerhalb eines Bedienfeldes verwendet werden können. Häufig benötigt man Auswahlfelder, die das Aktivieren oder Deaktivieren von Programmeinstellungen ermöglichen. XView stellt für diesen Zweck den Objekttyp `Panel_choice_item` zur Verfügung, dessen Erscheinungsbild und Funktion sich über verschiedene Attribute beeinflussen lässt. Das folgende Programm zeigt die verschiedenen Möglichkeiten, die mit diesem XView-Objekt realisiert werden können:

```
 1  /*
 2      auswahl.c - verschiedene Auswahlfelder erzeugen
 3  */
 4
 5  # include <stdio.h>
 6  # include <xview/frame.h>
 7  # include <xview/panel.h>
 8
 9  void quit_proc(Panel_item item, Event *event)
10   {
11    xv_destroy_safe(xv_get(item, PANEL_CLIENT_DATA));
```

Tabelle 8.4: Attribute für das Message-Objekt

Attribut	Parameter	Beschreibung
PANEL_ITEM_X	X-Position (Pixel)	Position innerhalb des Bedienfeldes
PANEL_ITEM_Y	Y-Position (Pixel)	Position innerhalb des Bedienfeldes
PANEL_LABEL_STRING	Zeichenkette	dargestellter Text
PANEL_LABEL_BOLD	TRUE oder FALSE	Fettschrift verwenden
PANEL_LABEL_X	X-Position (Pixel)	gleiche Bedeutung wie PANEL_ITEM_X
PANEL_LABEL_Y	Y-Position (Pixel)	gleiche Bedeutung wie PANEL_ITEM_Y
PANEL_ITEM_COLOR	Farbindex	Farbe des Textes
PANEL_NEXT_COL	Abstand (Pixel)	neue Spalte beginnen mit vorgegebenem Abstand (−1 für PANEL_ITEM_X_GAP)
PANEL_NEXT_ROW	Abstand (Pixel)	neue Zeile beginnen mit vorgegebenem Abstand (−1 für PANEL_ITEM_Y_GAP)

```
12    return;
13   }
14
15   void choice_proc(Panel_item item, int value,
16                    Event *event)
17    {
18     Frame frame;
19     static char string[100];
20
21     frame = xv_get(item, PANEL_CLIENT_DATA);
22     sprintf(string, "PANEL_VALUE=%d", value);
23     xv_set(frame, FRAME_LEFT_FOOTER, string, 0);
24     return;
25    }
26
27   int main(int argc, char *argv[])
28    {
```

```
29    Frame frame;
30    Panel panel;
31
32    xv_init(XV_INIT_ARGC_PTR_ARGV, &argc, argv, 0);
33
34    frame = xv_create(0, FRAME,
35      FRAME_LABEL,       "Auswahlfelder",
36      FRAME_SHOW_FOOTER, TRUE,
37      0);
38
39    panel = xv_create(frame, PANEL,
40      PANEL_LAYOUT, PANEL_VERTICAL,
41      0);
42
43    xv_create(panel, PANEL_BUTTON,
44      PANEL_LABEL_STRING, "Beenden",
45      PANEL_NOTIFY_PROC,  quit_proc,
46      PANEL_CLIENT_DATA,  frame,
47      0);
48
49    xv_create(panel, PANEL_CHOICE,
50      PANEL_LABEL_STRING,   "Auswahl1:",
51      PANEL_CHOICE_STRINGS, "Eins", "Zwei", NULL,
52      PANEL_NOTIFY_PROC,    choice_proc,
53      PANEL_CLIENT_DATA,    frame,
54      0);
55
56    xv_create(panel, PANEL_CHOICE,
57      PANEL_CHOOSE_ONE,     FALSE,
58      PANEL_LABEL_STRING,   "Auswahl2:",
59      PANEL_CHOICE_STRINGS, "Eins", "Zwei", NULL,
60      PANEL_NOTIFY_PROC,    choice_proc,
61      PANEL_CLIENT_DATA,    frame,
62      0);
63
64    xv_create(panel, PANEL_CHOICE,
65      PANEL_FEEDBACK,       PANEL_MARKED,
66      PANEL_LABEL_STRING,   "Auswahl3:",
67      PANEL_CHOICE_STRINGS, "Eins", "Zwei", NULL,
68      PANEL_NOTIFY_PROC,    choice_proc,
69      PANEL_CLIENT_DATA,    frame,
```

```
70      0);
71
72    xv_create(panel, PANEL_CHOICE,
73      PANEL_DISPLAY_LEVEL,  PANEL_CURRENT,
74      PANEL_LABEL_STRING,   "Auswahl4:",
75      PANEL_CHOICE_STRINGS, "Eins", "Zwei", NULL,
76      PANEL_NOTIFY_PROC,    choice_proc,
77      PANEL_CLIENT_DATA,    frame,
78      0);
79
80    xv_create(panel, PANEL_CHOICE,
81      PANEL_CHOOSE_ONE,     FALSE,
82      PANEL_FEEDBACK,       PANEL_MARKED,
83      PANEL_LABEL_STRING,   "Auswahl5:",
84      PANEL_CHOICE_STRINGS, "nur ein/aus", NULL,
85      PANEL_NOTIFY_PROC,    choice_proc,
86      PANEL_CLIENT_DATA,    frame,
87      0);
88
89    window_fit(panel);
90    window_fit(frame);
91    xv_main_loop(frame);
92    return(0);
93  }
```

Dieses Programm erzeugt das in Bild 8.5 dargestellte Fenster. Alle fünf Auswahlfelder rufen die gleiche Callback-Funktion `choice_proc()` (Zeile 15 bis 25) auf. Im Gegensatz zu den zuvor behandelten Schaltflächen besitzen Auswahlfelder einen „Wert“ (Attribut `PANEL_VALUE`), der als zweiter Parameter an die Callback-Funktion übergeben wird. Die Bedeutung dieses Wertes ist abhängig von den Attributen des Auswahlfeldes. Wenn beispielsweise `PANEL_CHOOSE_ONE` auf `TRUE` steht (Voreinstellung), enthält `PANEL_VALUE` den Index der Auswahl, angefangen bei 0 für die erste Auswahlmöglichkeit. Wird die Mehrfachauswahl ermöglicht, also `PANEL_CHOOSE_ONE` auf `FALSE` gestellt, so liefert `PANEL_VALUE` eine Zahl, bei der jedes Bit einer Auswahlmöglichkeit zugeordnet ist. Bit 0 repräsentiert die erste Auswahlmöglichkeit, Bit 1 die zweite und so weiter. Sind z.B. die erste und die dritte selektiert, so steht `PANEL_VALUE` auf $2^0 + 2^2 = 5$.

Hinweis: Bei Erzeugung eines Auswahlfeldes kann über das Attribut `PANEL_VALUE` auch ein Wert voreingestellt werden.

Bild 8.5: Verschiedene Auswahlfelder (oben), Auswahl 4 betätigt (unten)

Die folgenden Attribute lassen sich für Auswahlfelder einstellen (und teilweise kombinieren):

Attribut	Parameter	Beschreibung
`PANEL_CHOICE_STRINGS`	Liste mit Zeichenketten	Texte der möglichen Auswahlfelder
`PANEL_CHOOSE_ONE`	`TRUE` oder `FALSE`	immer nur ein Feld selektiert
`PANEL_FEEDBACK`	`PANEL_MARKED`	als Kästchen mit Haken darstellen
`PANEL_DISPLAY_LEVEL`	`PANEL_CURRENT`	als „Pull-Down“-Auswahl darstellen

8.2.9 Icons (Piktogramme)

Bisher haben wir für die Beschriftung von Bedienelementen immer Text verwendet. Natürlich bietet XView auch die Möglichkeit, Schaltflächen und andere Elemente mit Piktogrammen – also kleinen Grafiken – zu

„beschriften“. Der folgende Quelltext zeigt, wie eine Schaltfläche mit einer solchen Grafik versehen wird:

```
1  /*
2      button-icon.c - Schaltflaeche mit Grafik
3  */
4
5  # include <stdio.h>
6  # include <xview/frame.h>
7  # include <xview/panel.h>
8
9  void quit_proc(Panel_item item, Event *event)
10  {
11   xv_destroy_safe(xv_get(item, PANEL_CLIENT_DATA));
12   return;
13  }
14
15 int main(int argc, char *argv[])
16  {
17   Frame frame;
18   Panel panel;
19   Server_image button_image;
20   static unsigned short image_bits[16]
21     = {0x0180, 0x1998, 0x399c, 0x6186,
22        0x6186, 0xc183, 0xc183, 0xc183,
23        0xc183, 0xc003, 0xc003, 0x6006,
24        0x6006, 0x381c, 0x1ff8, 0x07e0};
25
26   xv_init(XV_INIT_ARGC_PTR_ARGV, &argc, argv, 0);
27
28   button_image = xv_create(0, SERVER_IMAGE,
29     XV_WIDTH,          16,
30     XV_HEIGHT,         16,
31     SERVER_IMAGE_BITS, image_bits,
32     0);
33
34   frame = xv_create(0, FRAME,
35     FRAME_LABEL, "Bedienelement mit Grafik",
36     0);
37
38   panel = xv_create(frame, PANEL, 0);
```

```
39
40    xv_create(panel, PANEL_MESSAGE,
41      PANEL_ITEM_Y,       36,
42      PANEL_LABEL_STRING, "Der Knopf zum Beenden:",
43      0);
44
45    xv_create(panel, PANEL_BUTTON,
46      PANEL_ITEM_Y,      30,
47      PANEL_LABEL_IMAGE, button_image,
48      PANEL_LABEL_WIDTH, 50,
49      PANEL_NOTIFY_PROC, quit_proc,
50      PANEL_CLIENT_DATA, frame,
51      0);
52
53    window_fit(panel);
54    window_fit(frame);
55    xv_main_loop(frame);
56    return(0);
57   }
```

In Zeile 19 wird die Variable `button_image` als XView-Objekt vom Typ `Server_image` definiert. Das hier verwendete Piktogramm hat eine Größe von 16×16 Pixel; die zugehörige Bitmap umfasst also 16 Worte à 2 Byte. Sie wird in den Zeilen 20 bis 24 als Feld aus 16 `Short`-Werten beschrieben und der Variablen `image_bits` zugewiesen. Das XView-Objekt für das Piktogramm wird in den Zeilen 28 bis 32 erzeugt. Neben der zuvor definierten Bitmap werden auch Breite und Höhe als Attribute angegeben. In Zeile 47 wird mit dem Attribut `PANEL_LABEL_IMAGE` dieses Objekt als „Beschriftung“ für die Schaltfläche eingesetzt. Das Ergebnis ist in Bild 8.6 dargestellt.

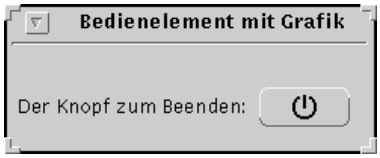

Bild 8.6: Schaltfläche mit Piktogramm

Natürlich muss man die Bitmap nicht wie hier in den Zeilen 20 bis 24 von Hand eingeben. Man kann sie mit einem Grafik-Programm wie `display`

(gehört zum Programmpaket „ImageMagick“) oder `xv` erstellen und als *Portable Bitmap* (PBM) speichern. Mit dem Programm `pbmtoicon` kann diese Datei dann in das Icon-Format konvertiert werden. In der Regel werden solche Icon-Dateien mit einer `include`-Anweisung in den Programmquelltext eingebunden.

Hinweis: Das Grafikprogramm `xpaint` eignet sich ebenfalls sehr gut, um Bitmap-Grafiken zu erstellen. Leider kann es nicht im PBM-Format speichern, sodass hier ein zusätzlicher Konvertierungsschritt notwendig wird.

Das Fenster-Piktogramm (Frame-Icon)

Der OpenLook Window-Manager `olwm` unterstützt ebenso wie der `fvwm` Programm-Icons. Anders als beim KDE, bei dem geschlossene (nicht beendete) Fenster durch eine Schaltfläche in der Task-Leiste repräsentiert werden, ersetzen diese Window-Manager solche Fenster durch Piktogramme (siehe auch Bild 8.1, linker Rand). Die `libxview` versieht Programme automatisch mit einem Default-Icon, das allerdings nur aus einem umrahmten, leeren Kasten besteht und damit nicht sehr aussagekräftig ist. Der folgende Quelltext zeigt, wie man eine andere Grafik einstellen kann:

```
 1  /*
 2      icon.c - Fenster mit Programm-Icon
 3  */
 4
 5  # include <stdio.h>
 6  # include <xview/frame.h>
 7  # include <xview/icon.h>
 8
 9  int main(int argc, char *argv[])
10   {
11    Frame frame;
12    Icon frame_icon;
13    Server_image icon_image, icon_mask_image;
14    static unsigned short image_bits[256] = {
15  # include <images/console.icon>
16      }, mask_image_bits[256] = {
17  # include <images/console_mask.icon>
18      };
19
```

```
20    xv_init(XV_INIT_ARGC_PTR_ARGV, &argc, argv, 0);
21
22    frame = xv_create(0, FRAME,
23      FRAME_LABEL,  "Fenster mit Icon",
24      FRAME_CLOSED, TRUE,
25      0);
26
27    icon_image = xv_create(0, SERVER_IMAGE,
28      XV_WIDTH,          64,
29      XV_HEIGHT,         64,
30      SERVER_IMAGE_BITS, image_bits,
31      0);
32
33    icon_mask_image = xv_create(0, SERVER_IMAGE,
34      XV_WIDTH,          64,
35      XV_HEIGHT,         64,
36      SERVER_IMAGE_BITS, mask_image_bits,
37      0);
38
39    frame_icon = xv_create(frame, ICON,
40      ICON_IMAGE,      icon_image,
41      ICON_MASK_IMAGE, icon_mask_image,
42      ICON_WIDTH,      64,
43      ICON_HEIGHT,     64,
44      ICON_LABEL,      "mein Icon",
45      0);
46
47    xv_set(frame, FRAME_ICON, frame_icon, 0);
48
49    xv_main_loop(frame);
50    return(0);
51  }
```

Zum Übersetzen dieses Quelltextes werden zwei Bitmap-Dateien im Icon-Format benötigt, wie sie mit dem Programm `pbmtoicon` erzeugt werden können (siehe oben). Diese Bitmap-Dateien müssen ein Format von 64×64 Pixel haben. Für das Beispielprogramm wurden zwei im Paket `xv32dev` enthaltene Dateien verwendet (Zeile 15 und 17). Damit erscheint beim Starten des Programms folgendes Icon (nicht beim KDE oder `fvwm95`):

Für ein Fenster-Piktogramm wird ein `Server_image` benötigt, wie es schon bei dem vorherigen Quelltext für die Grafik der Schaltfläche verwendet wurde. Dieses Image-Objekt wird in den Zeilen 27 bis 31 erzeugt und nutzt die Icon-Datei „`console.icon`“. Optional kann beim Fenster-Piktogramm zusätzlich eine weitere Bitmap als *Maske* verwendet werden. Sie dient als eine Art Schablone, die bestimmt, wo das Icon „durchsichtig“ ist, also die Hintergrundfarbe durchscheint. Das Image-Objekt `icon_mask_image` wird in den Zeilen 33 bis 37 erzeugt, wobei es als Bitmap die Datei „`console_mask.icon`“ verwendet.
In Zeile 39 bis 45 wird dann das eigentliche Piktogramm erzeugt und in Zeile 47 dem Fenster als Icon zugewiesen. Anstelle des Attributes `ICON_LABEL` kann die Beschriftung wahlweise auch mit dem Attribut `ICON_TRANSPARENT_LABEL` angegeben werden. In diesem Fall wird die Icon-Grafik im Bereich des Textes *nicht* ausgeblendet, Grafik und Text lassen sich dann mischen. Bei der im Beispiel verwendeten Bitmap ergibt sich dadurch kein Unterschied, da die Masken-Bitmap das Icon im Bereich der Schrift ausblendet.

8.2.10 Eingabefelder für Text und Zahlen

Neben Schaltflächen und Auswahlfeldern benötigt man häufig auch Eingabefelder. XView unterscheidet hier zwischen einem `TEXT_ITEM` für beliebigen Text (einzeilig) und einem `NUMERIC_TEXT_ITEM` für Zahlen. Texteingabefelder können z.B. zur Abfrage eines Dateinamens eingesetzt werden. Das folgende Programm erzeugt zwei Eingabefelder, eines für Text, das zweite für eine Zahl. Bild 8.7 zeigt das Erscheinungsbild dieser Felder.

```
1  /*
2      text.c - Eingabefelder fuer Text und Zahlen
3  */
4
5  # include <stdio.h>
6  # include <xview/frame.h>
```

```
 7  # include <xview/panel.h>
 8
 9  Panel_setting text_proc(Panel_item item, Event *event)
10   {
11    Frame frame;
12    char *label;
13    static char string[80];
14
15    frame = xv_get(item, PANEL_CLIENT_DATA);
16    label = (char *)xv_get(item, PANEL_LABEL_STRING);
17    if (strcmp(label, "Text:") == 0)
18      sprintf(string, "Text = '%s'",
19        (char *)xv_get(item, PANEL_VALUE));
20    else
21      sprintf(string, "Zahl = %d",
22        (int)xv_get(item, PANEL_VALUE));
23    xv_set(frame, FRAME_LEFT_FOOTER, string, 0);
24    return(PANEL_NEXT);
25   }
26
27  int main(int argc, char *argv[])
28   {
29    Frame frame;
30    Panel panel;
31
32    xv_init(XV_INIT_ARGC_PTR_ARGV, &argc, argv, 0);
33
34    frame = xv_create(0, FRAME,
35      FRAME_LABEL,       "Text & Zahlen",
36      FRAME_SHOW_FOOTER, TRUE,
37      0);
38
39    panel = xv_create(frame, PANEL,
40      PANEL_LAYOUT, PANEL_VERTICAL,
41      0);
42
43    xv_create(panel, PANEL_TEXT,
44      PANEL_LABEL_STRING,        "Text:",
45      PANEL_ITEM_Y,              10,
46      PANEL_VALUE_DISPLAY_WIDTH, 110,
47      PANEL_VALUE_STORED_LENGTH, 20,
```

```
48      PANEL_NOTIFY_PROC,          text_proc,
49      PANEL_CLIENT_DATA,          frame,
50      0);
51
52    xv_create(panel, PANEL_NUMERIC_TEXT,
53      PANEL_LABEL_STRING,         "Zahl:",
54      PANEL_VALUE_DISPLAY_WIDTH, 70,
55      PANEL_MIN_VALUE,            -10,
56      PANEL_MAX_VALUE,            20,
57      PANEL_NOTIFY_PROC,          text_proc,
58      PANEL_CLIENT_DATA,          frame,
59      0);
60
61    window_fit(panel);
62    window_fit(frame);
63    xv_main_loop(frame);
64    return(0);
65  }
```

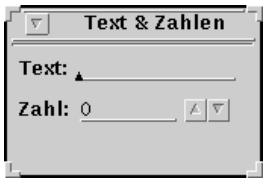

Bild 8.7: Eingabefelder für Text und Zahlen

Betrachtet man zunächst die Callback-Funktion für beide Eingabefelder in den Zeilen 9 bis 25, fällt auf, dass im Gegensatz zu bisher verwendeten Callback-Funktionen hier ein Rückgabewert vom Typ `Panel_setting` gesetzt wird. Dieser wird am Ende der Funktion in Zeile 24 auf `PANEL_NEXT` gesetzt, wodurch nach erfolgter Eingabe in einem der Felder der Cursor automatisch auf das andere Feld springt.
In Zeile 17 wird anhand der Beschriftung (Label) des Eingabefeldes festgestellt, ob die Funktion durch eine Eingabe im Text- oder im Zahlenfeld aufgerufen wurde. In beiden Fällen kann über das Attribut `PANEL_VALUE` der eingegebene Wert abgefragt werden – einmal als Zeichenkette (Zeile 19) und einmal als Integer-Zahl (Zeile 22).
Im Hauptprogramm werden Frame und Panel wie gehabt erzeugt, danach werden mit zwei weiteren `xv_create()`-Aufrufen das Text- und das

Zahleneingabefeld angelegt (Zeile 43 bis 50 und Zeile 52 bis 59). Folgende Attribute können für diese Elemente angegeben werden:

Attribut	Parameter	Beschreibung
PANEL_VALUE_DISPLAY_WIDTH	Breite (Pixel)	Breite des Eingabefeldes
PANEL_VALUE_STORED_LENGTH	Anz. Zeichen	max. Anzahl der Zeichen (nur Textfeld)
PANEL_VALUE_UNDERLINED	TRUE / FALSE	Eingabefeld unterstrichen
PANEL_MIN_VALUE	Zahl	Untergrenze (nur Zahlenfeld)
PANEL_MAX_VALUE	Zahl	Obergrenze (nur Zahlenfeld)
PANEL_MASK_CHAR	Zeichen	nur dieses Zeichen darstellen (z.B. für Passworteingabe)

8.2.11 Pegelanzeige und Schieberegler

Die Darstellung von Zahlenwerten ist manchmal anschaulicher über eine grafische Pegelanzeige, ähnlich der Aussteuerungsanzeige von Audioverstärkern. XView kennt solche Pegelanzeigen als Objekt vom Typ `Panel_gauge_item`. In gleicher Weise ist auch die Eingabe von Zahlen in bestimmten Fällen mit einem Schieberegler benutzerfreundlicher. Schieberegler sind in XView als `Panel_slider_item` verfügbar.
Das folgende Programm benutzt einen Schieberegler zum Einstellen eines Wertes zwischen 0 und 20. Über eine Pegelanzeige wird das Quadrat dieses Wertes dargestellt.

```
 1  /*
 2      slider.c - Schieberegler und Pegelanzeige
 3  */
 4
 5  # include <stdio.h>
 6  # include <xview/frame.h>
 7  # include <xview/panel.h>
 8
 9  void slider_proc(Panel_item item, int value,
10                   Event *event)
11   {
```

```
12    Panel_gauge_item gauge;
13
14    gauge = xv_get(item, PANEL_CLIENT_DATA);
15    xv_set(gauge, PANEL_VALUE, value*value, 0);
16    return;
17   }
18
19  int main(int argc, char *argv[])
20   {
21    Frame frame;
22    Panel panel;
23    Panel_slider_item slider;
24    Panel_gauge_item gauge;
25
26    xv_init(XV_INIT_ARGC_PTR_ARGV, &argc, argv, 0);
27
28    frame = xv_create(0, FRAME,
29      FRAME_LABEL,       "Schieberegler",
30      FRAME_SHOW_FOOTER, TRUE,
31      0);
32
33    panel = xv_create(frame, PANEL,
34      PANEL_LAYOUT, PANEL_VERTICAL,
35      0);
36
37    slider = xv_create(panel, PANEL_SLIDER,
38      PANEL_LABEL_STRING, "Wert:",
39      PANEL_MIN_VALUE,    0,
40      PANEL_MAX_VALUE,    20,
41      PANEL_TICKS,        5,
42      PANEL_SLIDER_WIDTH, 80,
43      PANEL_NOTIFY_PROC,  slider_proc,
44      0);
45
46    gauge = xv_create(panel, PANEL_GAUGE,
47      PANEL_LABEL_STRING, "Wert^2:",
48      PANEL_MIN_VALUE,    0,
49      PANEL_MAX_VALUE,    400,
50      PANEL_TICKS,        5,
51      PANEL_GAUGE_WIDTH,  120,
52      0);
```

```
53
54    xv_set(slider, PANEL_CLIENT_DATA, gauge, 0);
55
56    window_fit(panel);
57    window_fit(frame);
58    xv_main_loop(frame);
59    return(0);
60   }
```

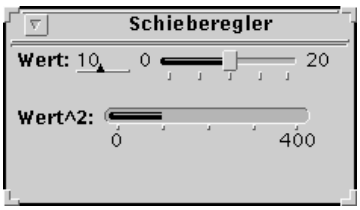

Bild 8.8: Schieberegler (Slider) und Pegelanzeige (Gauge)

Die XView-Objekte Slider und Gauge kennen eine ganze Reihe von Attributen. Tabelle 8.5 zeigt eine Übersicht.

8.2.12 Menüs

Wenn ein Programm eine Vielzahl von Funktionen unterstützt, wird dies allein mit Hilfe von Schaltflächen schnell unübersichtlich. Hier sollten stattdessen Menüs verwendet werden, die nur bei Bedarf „aufklappen“. XView unterscheidet verschiedene Arten von Menüs: *Pop-Up*-Menüs und *Pull-Down*-Menüs. Pop-Up-Menüs werden durch bestimmte Ereignisse wie z.B. das Drücken einer Maustaste innerhalb einer Zeichenfläche geöffnet, während Pull-Down-Menüs mit einer Schaltfläche verbunden sind. Das folgende Programm erzeugt ein typisches Pull-Down-Menü:

```
1  /*
2      menu.c - Programm mit Menue-Button
3  */
4
5  # include <stdio.h>
6  # include <xview/frame.h>
7  # include <xview/panel.h>
8  # include <xview/openmenu.h>
```

Tabelle 8.5: Attribute von Slider und Gauge

Attribut	Parameter	Beschreibung
PANEL_DIRECTION	PANEL_VERTICAL oder PANEL_HORIZONTAL	waagerecht oder senkrecht
PANEL_MIN_TICK_STRING	Zeichenkette	Beschriftung des kleinsten Wertes
PANEL_MAX_TICK_STRING	Zeichenkette	Beschriftung des größten Wertes
PANEL_TICKS	Anzahl	Anzahl der Unterteilungsstriche
PANEL_GAUGE_WIDTH PANEL_SLIDER_WIDTH	Länge (Pixel)	Ausdehnung
PANEL_SHOW_RANGE	TRUE oder FALSE	Min-/Max-Werte anzeigen
PANEL_MIN_VALUE_STRING	Zeichenkette	Text für Min-Wert
PANEL_MAX_VALUE_STRING	Zeichenkette	Text für Max-Wert
PANEL_SLIDER_END_BOXES	TRUE oder FALSE	Knopf für Min- und Max-Position

```
 9
10  # define KEY_FRAME 1
11
12  void menu_proc(Menu menu, Menu_item item)
13   {
14    Frame frame;
15    char *item_name;
16
17    frame = xv_get(menu, XV_KEY_DATA, KEY_FRAME);
18    item_name = (char *)xv_get(item, MENU_STRING);
19    xv_set(frame, FRAME_LEFT_FOOTER, item_name, 0);
20    return;
21   }
22
23  int main(int argc, char *argv[])
24   {
25    Frame frame;
26    Panel panel;
```

```
27    Menu menu;
28
29    xv_init(XV_INIT_ARGC_PTR_ARGV, &argc, argv, 0);
30
31    frame = xv_create(0, FRAME,
32      FRAME_LABEL,       argv[0],
33      FRAME_SHOW_FOOTER, TRUE,
34      0);
35
36    panel = xv_create(frame, PANEL, 0);
37
38    menu = xv_create(0, MENU,
39      MENU_STRINGS,     "Laden...", "Speichern...", NULL,
40      MENU_NOTIFY_PROC, menu_proc,
41      XV_KEY_DATA,      KEY_FRAME, frame,
42      0);
43
44    xv_create(panel, PANEL_BUTTON,
45      PANEL_LABEL_STRING, "Datei  ",
46      PANEL_ITEM_MENU,    menu,
47      0);
48
49    xv_main_loop(frame);
50    return(0);
51   }
```

Die Einrichtung des Pull-Down-Menüs erfolgt in zwei Schritten: Zuerst wird das Menü selbst erzeugt (Zeile 38 bis 42) und anschließend eine Schaltfläche, der dieses Menü mit dem Attribut `PANEL_ITEM_MENU` zugewiesen wird. Mit dem Attribut `MENU_STRINGS` für das Menü-Objekt wird eine mit `NULL` abgeschlossene Liste von Zeichenketten angegeben, die die Menüeinträge darstellen (Zeile 39). Für jeden Menüeintrag wird dabei ein eigenständiges XView-Objekt vom Typ `Menu_item` erzeugt. Wie bei den bisher beschriebenen Bedienelementen wird auch dem Menü eine Callback-Funktion (`NOTIFY_PROC`) zugewiesen – hier ist das die Funktion `menu_proc()`, die in den Zeilen 12 bis 21 definiert ist. Dieser Funktion werden beim Aufruf zwei Parameter übergeben: das Menü-Objekt und das Objekt des ausgewählten Menü-Eintrags. Bild 8.9 zeigt das Aussehen des aufgeklappten Menüs.
Bei der in dem Beispiel verwendeten Methode zur Erzeugung des Menüs wird für alle Menüeinträge eine gemeinsame Callback-Funktion ange-

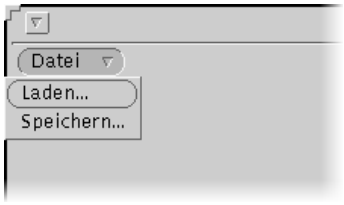

Bild 8.9: Geöffnetes Pull-Down-Menü

geben. Anhand des zweiten an diese Funktion übergebenen Parameters kann der vom Anwender gewählte Menüeintrag identifiziert werden. Häufig erfordern die Menüeinträge eines Menüs sehr unterschiedliche Aktionen (z.B. „Datei laden“, „Datei speichern“ und „Beenden“), sodass es sinnvoller ist, verschiedene Callback-Funktionen zu verwenden. Zu diesem Zweck können Menüeinträge *einzeln* definiert werden, Beispiel:

```
menu = xv_create(0, MENU,
  MENU_STRINGS,       "Laden...", "Speichern...", NULL,
  MENU_NOTIFY_PROC,   menu_proc,
  XV_KEY_DATA,        KEY_FRAME, frame,
  MENU_ITEM,
    MENU_STRING,      "Beenden",
    MENU_NOTIFY_PROC, quit_proc,
    0,
  0);
```

Hier wird über das Attribut `MENU_ITEM` ein weiterer Menüeintrag hinzugefügt. Es folgt eine Reihe von Attributen für diesen Menüeintrag (nicht für das Menü!), die mit einer 0 abgeschlossen ist. Danach könnten weitere Attribute für das Menü folgen oder – wie in diesem Fall – eine 0, um die Liste abzuschließen.
Wenn außer dem Text des Menüeintrags und der Callback-Funktion keine weiteren Angaben benötigt werden, so kann die Definition mit Hilfe des Menü-Attributes `MENU_ACTION_ITEM` abgekürzt werden:

```
menu = xv_create(0, MENU,
  MENU_STRINGS,     "Laden...", "Speichern...", NULL,
  MENU_NOTIFY_PROC, menu_proc,
  XV_KEY_DATA,      KEY_FRAME, frame,
  MENU_ACTION_ITEM, "Beenden", quit_proc,
  0);
```

Untermenüs

Enthält ein Menü sehr viele Einträge, so ist es für den Anwender häufig übersichtlicher, zusammengehörende Einträge in Untermenüs zusammenzufassen. XView bietet für diesen Zweck das Menü-Attribut `MENU_PULLRIGHT_ITEM`. Dieses Attribut benötigt zwei Parameter: den Namen des Menüeintrags und das Menü-Objekt, das als Untermenü eingebunden wird:

```
Menu menu, sub_menu;

sub_menu = xv_create(0, MENU,
  MENU_STRINGS,     "Laden...", "Speichern...", NULL,
  MENU_NOTIFY_PROC, files_proc,
  0);
menu = xv_create(0, MENU,
  MENU_PULLRIGHT_ITEM, "Datei", sub_menu,
  MENU_ACTION_ITEM,    "Beenden", quit_proc,
  0);
```

Bild 8.10 zeigt das zugehörige Erscheinungsbild, wenn Menü und Untermenü aufgeklappt werden.

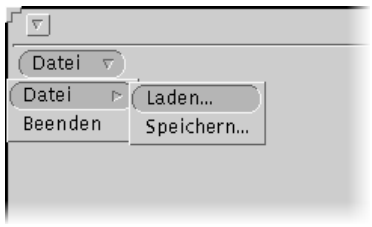

Bild 8.10: Pull-Down-Menü mit Untermenü

Zusätzliche Attribute

Es gibt noch einige nützliche Attribute für Menüs und Menüeinträge, die hier kurz erwähnt werden sollen:

Attribut	Parameter	Beschreibung
`MENU_INACTIVE`	`TRUE` oder `FALSE`	Menüpunkt gesperrt
`MENU_DEFAULT`	Nr. des Eintrags	bevorzugter Eintrag
`MENU_IMAGE`	Image-Objekt	Grafik statt Text
`MENU_FEEDBACK`	`TRUE` oder `FALSE`	Menüpunkt sichtbar

Die als Menütexte angegebenen Zeichenketten müssen Konstanten oder statische Variablen sein, da XView diese Zeichenketten nicht zwischenspeichert. Gegebenenfalls müssen die Texte erst in statische Variablen kopiert werden.

8.2.13 Command Frames

In den bisherigen Beispielprogrammen wurde immer ein Bedienfeld verwendet, welches das gesamte Fenster ausfüllte. Solche Fenster dienen in der Regel nicht als Hauptfenster, sondern werden z.B. geöffnet, um einen Dateinamen abzufragen oder einen Hilfetext darzustellen. Für diesen Zweck stellt XView einen speziellen Fenstertyp zur Verfügung: den *Command Frame*. Ein solches Fenster enthält automatisch ein Bedienfeld und besitzt – abhängig vom verwendeten Window-Manager – einen so genannten „Push Pin“. In der Regel werden solche Eingabe-Fenster schon zu Beginn des Programms erzeugt und im Programmverlauf über das Attribut `XV_SHOW` bei Bedarf dargestellt.
Das folgende Programm erzeugt ein Frame als „Hauptfenster“ und ein Command Frame (`FRAME_CMD`) zur Eingabe eines Dateinamens. Bei Betätigung der Schaltfläche „Datei speichern“ wird dieses Eingabefenster geöffnet.

```
 1  /*
 2      frame_cmd.c - Command Frame erzeugen
 3  */
 4
 5  # include <stdio.h>
 6  # include <xview/frame.h>
 7  # include <xview/panel.h>
 8
 9  void save_proc(Panel_item item, Event *event)
10   {
11    Frame frame_cmd;
12
13    frame_cmd = xv_get(item, PANEL_CLIENT_DATA);
14    xv_set(frame_cmd,
15      FRAME_CMD_PUSHPIN_IN, TRUE,
16      XV_SHOW,              TRUE,
17      0);
18    return;
19   }
20
```

```
21  void done_proc(Frame frame_cmd)
22   {
23    xv_set(frame_cmd, XV_SHOW, FALSE, 0);
24    return;
25   }
26
27  int main(int argc, char *argv[])
28   {
29    Frame frame, frame_cmd;
30    Panel panel, panel_cmd;
31
32    xv_init(XV_INIT_ARGC_PTR_ARGV, &argc, argv, 0);
33
34    frame = xv_create(0, FRAME,
35      FRAME_LABEL, "Hauptfenster",
36      0);
37
38    panel = xv_create(frame, PANEL, 0);
39
40    frame_cmd = xv_create(frame, FRAME_CMD,
41      FRAME_LABEL,     "Datei speichern",
42      FRAME_DONE_PROC, done_proc,
43      XV_SHOW,         FALSE,
44      0);
45
46    panel_cmd = xv_get(frame_cmd, FRAME_CMD_PANEL);
47
48    xv_create(panel_cmd, PANEL_TEXT,
49      PANEL_LABEL_STRING,        "Dateiname",
50      PANEL_VALUE_DISPLAY_WIDTH, 120,
51      PANEL_ITEM_Y,              20,
52      0);
53
54    window_fit(panel_cmd);
55    window_fit(frame_cmd);
56
57    xv_create(panel, PANEL_BUTTON,
58      PANEL_LABEL_STRING, "Datei speichern",
59      PANEL_NOTIFY_PROC,  save_proc,
60      PANEL_CLIENT_DATA,  frame_cmd,
61      0);
```

```
62
63     xv_main_loop(frame);
64     return(0);
65   }
```

Bild 8.11 zeigt das Hauptfenster und das eingeblendete Eingabefenster. Betrachten wir zunächst das Hauptprogramm. In den Zeilen 34 bis 38

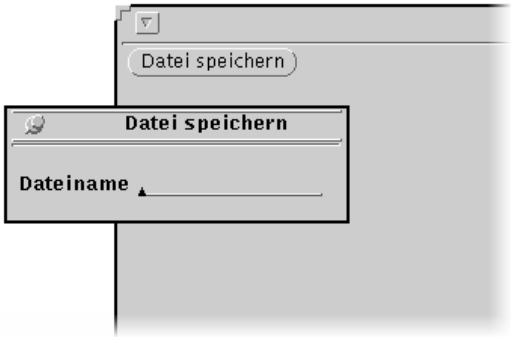

Bild 8.11: Hauptfenster mit geöffnetem *Command Frame*

wird zunächst wie gehabt das Hauptfenster und ein zugehöriges Bedienfeld erzeugt. In den Zeilen 40 bis 44 folgt das Erzeugen des Command Frames. Die Attribute `XV_SHOW` und `FRAME_DONE_PROC` bewirken, dass dieses Eingabefenster zunächst nicht sichtbar ist und dass die Callback-Funktion `done_proc()` beim Schließen des Command Frames aufgerufen wird.[7] Um nun ein Bedienelement auf dem Eingabefenster zu platzieren, muss das Bedienfeld dieses Fensters bekannt sein.[8] In Zeile 46 wird es mit `xv_get()` abgefragt. Damit kann in den Zeilen 48 bis 52 dann das Bedienelement (Eingabefeld für den Dateinamen) auf dem Command Frame erzeugt werden. Zuletzt wird in Zeile 57 bis 61 die Schaltfläche zum Öffnen des Eingabefensters erzeugt.

8.2.14 Hinweis- und Abfragefenster

Mit den bisher vorgestellten XView-Objekten lässt sich in ähnlicher Weise wie die zuvor beschriebenen Eingabefenster auch ein Hinweisfenster –

[7] Das Attribut `FRAME_DONE_PROC` ist nur bei Command Frames wirksam.
[8] Da Command Frames automatisch ein Bedienfeld haben, muss dieses nicht extra erzeugt werden.

z.B. für eine Sicherheitsabfrage beim Beenden des Programms – realisieren. XView stellt für diese Art Fenster spezielle Funktionen und Objekte zur Verfügung, die dem Programmierer die Arbeit etwas erleichtern.
Eine sehr einfache Möglichkeit zur Erzeugung eines Hinweisfensters bietet die Funktion `notice_prompt()`:

```
int notice_prompt(Xv_Window window,
                  Event *return_event, ...);
```

Die Funktion erzeugt ein Hinweisfenster, das beliebigen Text und mehrere Schaltflächen enthalten kann. Im Gegensatz zu `xv_create()` erzeugt `notice_prompt()` nicht nur ein Objekt, sondern hält die Ausführung des Programms so lange an, bis der Anwender eine Schaltfläche in dem Hinweisfenster betätigt. Als ersten Parameter erwartet die Funktion ein (Fenster-)Objekt, dem das Hinweisfenster zugeordnet wird. Als zweiter Parameter kann der Zeiger auf eine Variable vom Typ `Event` angegeben werden. `notice_prompt()` trägt dann in diese Variable das Ereignis ein, mit dem das Hinweisfenster geschlossen wird – in der Regel ist das ein Mausklick auf eine der Schaltflächen. Wird diese Information nicht benötigt, kann stattdessen der zweite Parameter zu `NULL` gesetzt werden. Die weiteren Parameter können beliebige Attribute des Notice-Objektes sein, mit denen der Hinweistext und die Schaltflächen bestimmt werden. Als Rückgabewert liefert das Programm eine Integer-Zahl, die angibt, welche Schaltfläche betätigt wurde. Das folgende Programm benutzt die Funktion `notice_prompt()`, um eine Sicherheitsabfrage beim Beenden des Programms zu realisieren:

```
 1  /*
 2      notice.c - Eine Warnung ausgeben
 3  */
 4
 5  # include <stdio.h>
 6  # include <xview/frame.h>
 7  # include <xview/panel.h>
 8  # include <xview/notice.h>
 9
10  void quit_proc(Panel_item item, Event *event)
11   {
12    Frame frame;
13    int choice;
14
15    frame = xv_get(item, PANEL_CLIENT_DATA);
```

```
16    choice = notice_prompt(frame, NULL,
17      NOTICE_MESSAGE_STRING, "Warnung!\n"
18        "Nicht gespeicherte Daten gehen verloren!\n"
19        "Wollen Sie wirklich beenden?",
20      NOTICE_BUTTON_YES,     "  Ja  ",
21      NOTICE_BUTTON_NO,      " Nein ",
22      0);
23    if (choice == NOTICE_YES)
24      xv_destroy_safe(frame);
25    return;
26   }
27
28  int main(int argc, char *argv[])
29   {
30    Frame frame;
31    Panel panel;
32
33    xv_init(XV_INIT_ARGC_PTR_ARGV, &argc, argv, 0);
34
35    frame = xv_create(0, FRAME,
36      FRAME_LABEL, argv[0],
37      0);
38
39    panel = xv_create(frame, PANEL, 0);
40
41    xv_create(panel, PANEL_BUTTON,
42      PANEL_LABEL_STRING, "Beenden",
43      PANEL_NOTIFY_PROC,  quit_proc,
44      PANEL_CLIENT_DATA,  frame,
45      0);
46
47    xv_main_loop(frame);
48    return(0);
49   }
```

Im Hauptprogramm (Zeile 28 bis 49) wird wie bisher ein Fenster mit Bedienfeld und einer Schaltfläche erzeugt. Die Callback-Funktion `quit_proc()`, die mit dieser Schaltfläche verknüpft ist, ist in den Zeilen 10 bis 26 definiert, der Aufruf der Funktion `notice_prompt()` erfolgt in den Zeilen 16 bis 22. Über das Attribut `NOTICE_MESSAGE_STRING` wird der Hinweistext als Zeichenkette angegeben. Ferner werden mit den Attributen `NOTICE_BUTTON_YES` und `NOTICE_BUTTON_NO` die Texte für die

Ja/Nein-Schaltflächen festegelegt. Bild 8.12 zeigt das Erscheinungsbild des Hinweisfensters.

Bild 8.12: Warnhinweis erzeugt mit `notice_prompt()`

Klickt der Anwender die Schaltfläche „Ja“, liefert `notice_prompt()` den Wert `NOTICE_YES` zurück; bei Betätigung der Schaltfläche „Nein“ ist der Rückgabewert `NOTICE_NO`. Alternativ zu den Standard-Schaltflächen (Ja/Nein) können beliebig viele andere Schaltflächen angegeben werden. Dies geschieht mit dem Attribut `NOTICE_BUTTON`, gefolgt von dem Text der Schaltfläche und einer Integer-Zahl, die bei Betätigung dieser Fläche zurückgegeben wird:

```
choice = notice_prompt(frame, NULL,
  NOTICE_MESSAGE_STRING, "Bitte hier klicken.",
  NOTICE_BUTTON,         "Danke!", 100,
  0);
```

XView bietet eine weitere Möglichkeit für ein Hinweisfenster. Dazu wird das Notice-Objekt mit `xv_create()` erzeugt, wobei die gleichen Attribute wie bei der Funktion `notice_prompt()` angegeben werden können. Das Erzeugen des Fensters kann hier zu Beginn des Programms erfolgen, geöffnet wird es erst durch Setzen des Attributs `XV_SHOW` auf `TRUE`. Das folgende Programm zeigt diese Möglichkeit auf:

```
1  /*
2      notice2.c - Eine Warnung ausgeben
3  */
```

```
 4
 5  # include <stdio.h>
 6  # include <xview/frame.h>
 7  # include <xview/panel.h>
 8  # include <xview/notice.h>
 9
10  # define KEY_FRAME 1
11  # define KEY_NOTICE 2
12
13  void quit_proc(Panel_item item, Event *event)
14   {
15    Frame frame;
16    Xv_notice notice;
17    int choice;
18
19    frame = xv_get(item, XV_KEY_DATA, KEY_FRAME);
20    notice = xv_get(item, XV_KEY_DATA, KEY_NOTICE);
21    xv_set(notice,
22      XV_SHOW,       TRUE,
23      NOTICE_STATUS, &choice,
24      0);
25    if (choice == NOTICE_YES)
26      xv_destroy_safe(frame);
27    return;
28   }
29
30  int main(int argc, char *argv[])
31   {
32    Frame frame;
33    Panel panel;
34    Xv_notice notice;
35
36    xv_init(XV_INIT_ARGC_PTR_ARGV, &argc, argv, 0);
37
38    frame = xv_create(0, FRAME,
39      FRAME_LABEL, argv[0],
40      0);
41
42    panel = xv_create(frame, PANEL, 0);
43
44    notice = xv_create(frame, NOTICE,
```

```
45      NOTICE_MESSAGE_STRING, "Warnung!\n"
46        "Nicht gespeicherte Daten gehen verloren!\n"
47        "Wollen Sie wirklich beenden?",
48      NOTICE_BUTTON_YES,     "  Ja  ",
49      NOTICE_BUTTON_NO,      " Nein ",
50      0);
51
52    xv_create(panel, PANEL_BUTTON,
53      PANEL_LABEL_STRING, "Beenden",
54      PANEL_NOTIFY_PROC,  quit_proc,
55      XV_KEY_DATA,        KEY_FRAME, frame,
56      XV_KEY_DATA,        KEY_NOTICE, notice,
57      0);
58
59    xv_main_loop(frame);
60    return(0);
61  }
```

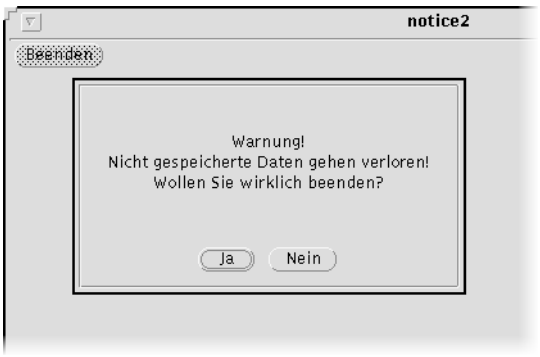

Bild 8.13: Warnhinweis erzeugt mit `xv_create()`

8.2.15 Farben einrichten

In den vorangegangenen Abschnitten ist bereits das Attribut `PANEL_ITEM_COLOR` vorgestellt worden, mit dem sich die Farbe einer Schaltfläche oder eines anderen Bedienelementes ändern lässt. Als Farben standen jedoch bisher nur die Standardfarben von XView zur Verfügung: Schwarz, Weiß und verschiedene Grautöne. Natürlich können auch andere Farb-

werte eingestellt werden. Dazu muss zunächst ein so genanntes *Color Map Segment* (CMS) erzeugt werden. Damit können neue Farben angelegt werden:

```
CMS cms;

cms = xv_create(0, CMS,
  CMS_SIZE          3,
  CMS_NAMED_COLORS, "red", "white", "blue", NULL,
  0);
```

Man kann die Farben mit Namen angeben oder – mit einem „#“ vorangestellt – als hexadezimale RGB[9]-Werte, z.B. „#c0a080“. Mit dem Attribut `WIN_CMS` lässt sich ein solches Farbsegment beispielsweise einem `Frame` oder einem `Panel` zuorden. Bei Verwendung für ein `Panel` sollten jedoch auch die Standardfarben enthalten sein, um die 3D-Effekte an Rahmen und Schaltflächen beizubehalten. Die Konstante `CMS_CONTROL_COLORS` gibt die Anzahl dieser Standardfarben an und mit dem Attribut `CMS_CONTROL_CMS` können sie automatisch in das Farbsegment integriert werden:

```
CMS cms;

cms = xv_create(0, CMS,
  CMS_SIZE          CMS_CONTROL_COLORS+3,
  CMS_CONTROL_CMS,  TRUE,
  CMS_NAMED_COLORS, "red", "white", "blue", NULL,
  0);
```

Die neu angegebenen Farben – hier Rot, Weiß und Blau – werden in dem Farbsegment *hinter* den Standardfarben eingereiht; der Index für die erste neue Farbe ist demnach `CMS_CONTROL_COLORS+0`.
Das folgende Programm erzeugt ein Fenster und ein Bedienfeld. Diesem Bedienfeld wird ein neues Farbsegment zugewiesen, dessen Farben für ein Message- und ein Button-Objekt verwendet werden:

```
1  /*
2      color.c - Eine neue Farbe einrichten
3  */
4
5  # include <stdio.h>
```

[9] „RGB“ steht für „Rot-Grün-Blau“.

```
 6  # include <xview/frame.h>
 7  # include <xview/panel.h>
 8  # include <xview/cms.h>
 9
10  void quit_proc(Panel_item item, Event *event)
11   {
12    xv_destroy_safe(xv_get(item, PANEL_CLIENT_DATA));
13    return;
14   }
15
16  int main(int argc, char *argv[])
17   {
18    Frame frame;
19    Panel panel;
20    Cms cms;      /* Color Map Segment */
21
22    xv_init(XV_INIT_ARGC_PTR_ARGV, &argc, argv, 0);
23
24    cms = xv_create(0, CMS,
25      CMS_SIZE,         CMS_CONTROL_COLORS+2,
26      CMS_CONTROL_CMS,  TRUE,
27      CMS_NAMED_COLORS, "red", "blue", NULL,
28      0);
29    if (cms == XV_NULL)
30     {
31      fprintf(stderr, "color: Can't create CMS.\n");
32      return(1);
33     }
34
35    frame = xv_create(0, FRAME,
36      FRAME_LABEL, "Farbdemo",
37      XV_WIDTH,    300,
38      XV_HEIGHT,   100,
39      0);
40
41    panel = xv_create(frame, PANEL,
42      PANEL_BORDER, TRUE,
43      WIN_CMS,      cms,
44      0);
45
46    xv_create(panel, PANEL_MESSAGE,
```

```
47      PANEL_LABEL_STRING, "Jetzt auch in Farbe:",
48      PANEL_ITEM_COLOR,   CMS_CONTROL_COLORS+0,
49      PANEL_ITEM_Y,       20,
50      0);
51
52    xv_create(panel, PANEL_BUTTON,
53      PANEL_LABEL_STRING, "Wirklich beenden?",
54      PANEL_ITEM_COLOR,   CMS_CONTROL_COLORS+1,
55      PANEL_NOTIFY_PROC,  quit_proc,
56      PANEL_CLIENT_DATA,  frame,
57      PANEL_ITEM_Y,       18,
58      0);
59
60    xv_main_loop(frame);
61    return(0);
62   }
```

Man beachte die Abfrage in Zeile 29. Wird der X11-Server mit geringer Farbzahl (z.B. 16 Farben) gestartet, so kann es durchaus vorkommen, dass schon alle Farben „belegt“ sind. In diesem Fall scheitert das Reservieren neuer Farben, was die XView-Bibliothek mit dem Rückgabewert `XV_NULL` signalisiert.

8.2.16 Zeichenflächen

Die bisher vorgestellen XView-Objekte nutzten das X-Window-System nur für die Gestaltung von Bedienelementen und deren Bedienung mit der Maus. „Echte“ Grafikfunktionen wie das Zeichnen von Linien sind damit nicht möglich. Zum Zeichnen oder Malen benötigen wir eine Zeichenfläche, das entsprechende XView-Objekt heißt *Canvas* (englisch für „Leinwand“). Auf einer solchen Zeichenfläche können sämtliche Zeichenfunktionen der Xlib, also der Basis-Bibliothek des X11-Systems angewendet werden, z.B. für Linien, Polygone, Ellipsen und natürlich auch Text.
Die Erzeugung und Handhabung eines Canvas-Objektes lässt sich am besten anhand eines Beispielprogramms erläutern. Das folgende Programm erzeugt eine Zeichenfläche und wendet verschiedene Zeichenfunktionen darauf an (siehe Bild 8.14):

```
1  /*
2      canvas.c - Eine Zeichenflaeche
3  */
```

```
 4
 5  # include <stdio.h>
 6  # include <xview/frame.h>
 7  # include <xview/cms.h>
 8  # include <xview/canvas.h>
 9  # include <xview/xv_xrect.h>
10  # include <xview/font.h>
11
12  # define KEY_FONT 1
13
14  void repaint_proc(Canvas canvas, Xv_Window paint_win,
15    Display *display, Window xwin, Xv_xrectlist *areas)
16   {
17    GC gc;
18    XGCValues gc_values;
19    Cms cms;
20    int width, height;
21    long red, blue;
22    Xv_Font font;
23
24    width = xv_get(paint_win, XV_WIDTH);
25    height = xv_get(paint_win, XV_HEIGHT);
26    cms = xv_get(canvas, WIN_CMS);
27    red = xv_get(cms, CMS_PIXEL, 1);
28    blue = xv_get(cms, CMS_PIXEL, 2);
29    font = xv_get(canvas, XV_KEY_DATA, KEY_FONT);
30    gc = XCreateGC(display, xwin, 0, &gc_values);
31
32    XClearWindow(display, xwin);
33
34    XDrawLine(display, xwin, gc, 10, 10, 200, 80);
35
36    gc_values.foreground = red;
37    gc_values.line_width = 3;
38    XChangeGC(display, gc, GCForeground | GCLineWidth,
39      &gc_values);
40
41    XDrawArc(display, xwin, gc, 1, 1,
42      width-3, height-3, 0, 360*64);
43
44    XSetForeground(display, gc, blue);
```

```
45    if (font != XV_NULL)
46      XSetFont(display, gc, xv_get(font, XV_XID));
47
48    XDrawString(display, xwin, gc, 20, 100,
49      "Hier ist eine Leinwand", 22);
50
51    XFreeGC(display, gc);
52    return;
53   }
54
55  int main(int argc, char *argv[])
56   {
57    Frame frame;
58    Cms cms;
59    Xv_Font font;
60
61    xv_init(XV_INIT_ARGC_PTR_ARGV, &argc, argv, 0);
62
63    frame = xv_create(0, FRAME,
64      FRAME_LABEL, argv[0],
65      XV_WIDTH,    300,
66      XV_HEIGHT,   200,
67      0);
68
69    cms = xv_create(0, CMS,
70      CMS_SIZE,         4,
71      CMS_NAMED_COLORS, "white", "red", "blue",
72        "black", NULL,
73      0);
74    if (cms == XV_NULL)
75     {
76      fprintf(stderr, "color: Can't create CMS.\n");
77      return(1);
78     }
79
80    font = xv_find(frame, FONT,
81      FONT_FAMILY, FONT_FAMILY_GALLENT,
82      FONT_STYLE,  FONT_STYLE_ITALIC,
83      FONT_SIZE,   24,
84      0);
85
```

```
86    xv_create(frame, CANVAS,
87      CANVAS_REPAINT_PROC,   repaint_proc,
88      CANVAS_X_PAINT_WINDOW, TRUE,
89      CANVAS_RETAINED,       FALSE,
90      WIN_CMS,               cms,
91      XV_KEY_DATA,           KEY_FONT, font,
92      0);
93
94    xv_main_loop(frame);
95    return(0);
96   }
```

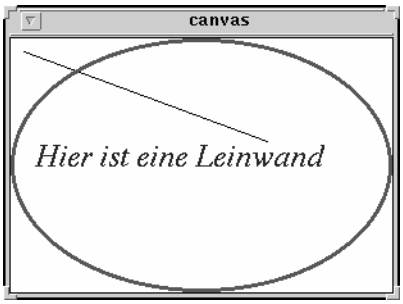

Bild 8.14: Eine Zeichenfläche mit Text und Grafik

Zunächst sei das Hauptprogramm (Zeile 55 bis 96) betrachtet. Dort werden zuerst der Frame und ein Farbsegment (CMS) erzeugt. In den Zeilen 80 bis 84 wird dann eine bestimmte Schriftart (*Font*) „gesucht“. Alternativ kann man statt der Schriftartfamilie, dem Stil und der Größe den Zeichensatznamen auch direkt angeben:

```
FONT_NAME, "times-italic-24",
```

Als letztes Objekt wird die Zeichenfläche selbst erzeugt (Zeile 86 bis 92). Mit dem Attribut `CANVAS_REPAINT_PROC` legt man die Callback-Funktion zum Zeichnen des Inhalts fest. Alle Ausgaben in der Zeichenfläche sollten innerhalb dieser Funktion erfolgen. Dadurch wird jederzeit die korrekte Darstellung der Zeichenfläche gewährleistet. Das Attribut `CANVAS_X_PAINT_WINDOW` legt fest, welche Form – genauer gesagt, welche Parameter – die Callback-Funktion hat. Wenn Xlib-Zeichenfunktionen zur Anwendung kommen, bietet sich hier die Einstellung `TRUE` an, da in

diesem Fall bestimmte, für diese Zeichenfunktionen relevante Parameter übergeben werden.

Hat `CANVAS_X_PAINT_WINDOW` den falschen Wert, passen die Parameter nicht zu der Callback-Funktion und es kommt eventuell zu einer Speicherzugriffsverletzung (Segmentation Fault).

Das Attribut `CANVAS_RETAINED` gibt an, ob die Zeichenfläche ihren Inhalt sichert. Wird dieses Attribut auf `TRUE` gesetzt, merkt sich die Zeichenfläche beim Verkleinern des Fensters den ursprünglich sichtbaren Inhalt und stellt diesen automatisch wieder dar, wenn das Fenster auf die vorherige Größe erweitert wird. Da in dem Beispielprogramm eine Ellipse gezeichnet wird, die sich der Größe der Zeichenfläche anpasst, soll hier die Zeichenfläche bei jedem Vergrößern des Fensters vollständig neu gezeichnet werden. Deshalb stellt man `CANVAS_RETAINED` in diesem Fall auf `FALSE`.
Die Callback-Funktion `repaint_proc()` zum Zeichnen des Canvas-Inhalts ist in den Zeilen 14 bis 53 definiert. Neben dem Objekt der neu zu zeichnenden Zeichenfläche bekommt diese Funktion eine ganze Reihe weiterer Parameter übergeben. Dabei sind `display` und `xwin` zwei für die meisten Xlib-Zeichenfunktionen erforderliche Variablen. Die genaue Bedeutung der einzelnen Parameter soll hier nicht weiter erläuterter werden – für die Nutzung der Xlib-Funktionen ist das nicht unbedingt erforderlich. Es sei nur der Hinweis gegeben, dass der Parameter `areas` (sofern $\neq$ `NULL`) auf eine Liste der neu zu zeichnenden Bereiche verweist. Damit ließe sich der Bereich der Zeichenfläche, der wiederhergestellt wird, auf das Notwendigste begrenzen.
In Zeile 17 wird eine Variable vom Typ `GC` definiert und in Zeile 30 initialisiert. Dabei steht das Kürzel GC für *Graphic Context*. Dieses Objekt beschreibt die aktuellen Einstellungen für Zeichenfunktionen: Vordergrund- und Hintergrundfarbe, Linienstärke und -stil, Füllstil und Schriftart. Fast alle Xlib-Zeichenfunktionen benötigen das GC-Objekt. Erzeugt wird das GC-Objekt mit Hilfe von `XCreateGC()`:

```
GC XCreateGC(Display *display, Drawable d,
        unsigned long value_mask, XGCValues *values);
```

Als „`Drawable`“ (*zeichenfähig*) wird hier das X11-Window der Zeichenfläche (`xwin`) angegeben. Der dritte Parameter stellt eine Bit-Maske dar, die angibt, welche Attribute des zu erzeugenden GC initialisiert werden sollen. Als vierter Parameter muss der Zeiger auf eine Struktur vom Typ `XGCValues` angegeben werden, die für jedes GC-Attribut ein Element enthält. Dieses Prinzip der Initialisierung wird im Zusammenhang mit der Funktion `XChangeGC()` weiter unten erläutert. Beim Erzeugen des

GC-Objektes ist hier als Bit-Maske 0 angegeben, sodass für alle GC-Parameter die voreingestellten Werte verwendet werden.
Als erste Grafikfunktion wird in Zeile 32 zunächst die Zeichenfläche gelöscht. Das ist in diesem Fall erforderlich, da sich der Inhalt der Zeichenfläche beim Verändern der Fenstergröße ändern soll, sodass dabei zunächst der alte Inhalt gelöscht werden muss. Danach wird mit `XDrawLine()` eine Linie gezeichnet. In den Zeilen 36 bis 39 wird dann der Graphic Context verändert: Die Zeichenfarbe wird jetzt auf Rot und die Strichstärke auf 3 Pixel gestellt. Dies geschieht mit der Funktion `XChangeGC()`, wobei zunächst die entsprechenden Einträge der Variablen `gc_values` modifiziert werden und genau diese Einträge mit `XChangeGC()` in das GC-Objekt übernommen werden. Dazu ist als Bit-Maske die Oder-Verknüpfung aus `GCForeground` und `GCLineWidth` angegeben.
Nach Umstellen der Zeichenfarbe und der Linienstärke erfolgt mit `XDrawArc()` das Zeichnen einer Ellipse. Die beiden letzten Parameter geben den Anfangs- und Endwinkel in 1/64°-Schritten an. Damit ist auch das Zeichnen von Kreissegmenten möglich. In den Zeilen 44 bis 46 wird dann die Zeichenfarbe und die Schriftart für Text neu eingestellt. Auch dieser Schritt könnte wie zuvor mit `XChangeGC()` erfolgen. Man verwendet hier jedoch eine einfachere Alternative, nämlich die Funktionen `XSetForeground()` und `XSetFont()`.
Mit dem `XDrawString()`-Aufruf in Zeile 48 und 49 wird dann zuletzt der Text an die Position (20,100) geschrieben. Vor Verlassen der Callback-Funktion wird das anfangs erzeugte GC-Objekt wieder freigegeben (Zeile 51).
Alle Grafikfunktionen der Xlib zu beschreiben, würde den Rahmen dieses Buches bei Weitem sprengen. Hier sei auf die ausführliche Dokumentation in den „`man`“-Hilfeseiten verwiesen.

Hinweis: Neben dem Canvas-Attribut `CANVAS_REPAINT_PROC` steht auch das Attribut `CANVAS_RESIZE_PROC` zur Verfügung, mit dem eine Funktion angegeben werden kann, die bei Veränderung der Größe der Zeichenfläche aufgerufen wird:

```
void resize_proc(Canvas canvas, int width, int height)
```

Wird die Zeichenfläche vom Anwender vergrößert (durch Vergrößern des Fensters), so wird in der Regel zuerst die als `RESIZE_PROC` angegebene Funktion aufgerufen und anschließend die mit dem Attribut `REPAINT_PROC` verbundene Funktion.

Zeichenflächen mit Rollbalken

Bei dem vorangegangenen Beispielprogramm wird die Größe der Zeichenfläche der Fenstergröße angepasst – sowohl beim Erzeugen der Zeichenfläche als auch später, wenn der Anwender die Fenstergröße verändert. Mit den Canvas-Attributen `CANVAS_WIDTH` und `CANVAS_HEIGHT` kann die Größe aber auch explizit vorgegeben werden. Über die Attribute `CANVAS_AUTO_SHRINK` und `CANVAS_AUTO_EXPAND` kann zusätzlich eingestellt werden, ob sich die Größe der Zeichenfläche nachträglich durch Ändern der Fenstergröße modifizieren lässt. Ist das Fenster kleiner als die Zeichenfläche, kann mit Hilfe von Rollbalken (*Scrollbar*) der sichtbare Ausschnitt gewählt werden.
Das folgende Programm erzeugt eine Zeichenfläche mit einer festen Größe von 800×800 Punkten, die mit einem vertikalen und einem horizontalen Rollbalken versehen ist (Bild 8.15).

```
1  /*
2      canvas2.c - Zeichenflaeche mit Rollbalken
3  */
4
5  # include <stdio.h>
6  # include <xview/frame.h>
7  # include <xview/canvas.h>
8  # include <xview/scrollbar.h>
9  # include <xview/xv_xrect.h>
10
11 void repaint_proc(Canvas canvas, Xv_Window paint_win,
12   Display *display, Window xwin, Xv_xrectlist *areas)
13  {
14   int i;
15   GC gc;
16
17   gc = DefaultGC(display, DefaultScreen(display));
18
19   XClearWindow(display, xwin);
20
21   for (i=0; i<=80; i++)
22     XDrawLine(display, xwin, gc, i*10, 0, 800-i*10,
23               800);
24   return;
25  }
26
```

```
27  int main(int argc, char *argv[])
28   {
29    Frame frame;
30    Canvas canvas;
31
32    xv_init(XV_INIT_ARGC_PTR_ARGV, &argc, argv, 0);
33
34    frame = xv_create(0, FRAME,
35      FRAME_LABEL, argv[0],
36      XV_WIDTH,    400,
37      XV_HEIGHT,   300,
38      0);
39
40    canvas = xv_create(frame, CANVAS,
41      CANVAS_WIDTH,          800,
42      CANVAS_HEIGHT,         800,
43      CANVAS_AUTO_SHRINK,    FALSE,
44      CANVAS_AUTO_EXPAND,    FALSE,
45      CANVAS_REPAINT_PROC,   repaint_proc,
46      CANVAS_X_PAINT_WINDOW, TRUE,
47      0);
48
49    xv_create(canvas, SCROLLBAR,
50      SCROLLBAR_DIRECTION, SCROLLBAR_HORIZONTAL,
51      0);
52
53    xv_create(canvas, SCROLLBAR,
54      SCROLLBAR_DIRECTION, SCROLLBAR_VERTICAL,
55      0);
56
57    xv_main_loop(frame);
58    return(0);
59   }
```

In dem Hauptprogramm ab Zeile 27 wird wie zuvor erst das Fenster und dann die Zeichenfläche erzeugt. Anschließend werden in den Zeilen 49 bis 55 zusätzlich zwei XView-Objekte vom Typ Scrollbar erzeugt, die über den ersten Parameter der Funktion `xv_create()` mit der Zeichenfläche verknüpft sind.

Hinweis: Der für die Zeichenfunktionen erforderliche *Graphic Context* (GC) wird hier – anders als beim voherigen Beispielprogramm – nicht

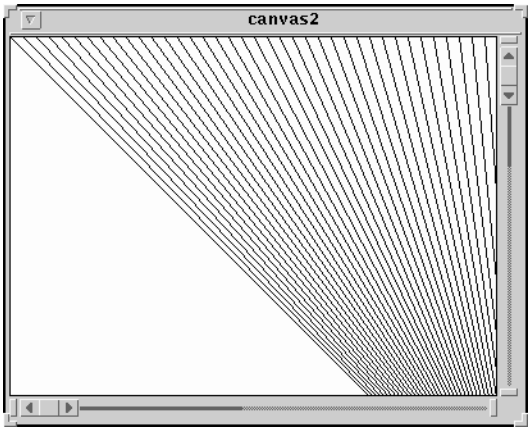

Bild 8.15: Zeichenfläche mit Rollbalken

neu erzeugt. Stattdessen wird das Standard-GC-Object verwendet, das über die Funktion `DefaultGC()` verfügbar ist (Zeile 17).

Bitmap-Grafiken darstellen

Ein weiteres Beispiel zum Thema Zeichenflächen soll demonstrieren, wie man Bitmap-Grafiken (z.B. Fotos) mit Hilfe von XView darstellen kann. Bei einer Zeichenfläche handelt es sich um ein XView-Objekt, das keinen direkten Zugriff auf den Bildspeicher zur Verfügung stellt. Auch hier bedient man sich wieder geeigneter Funktionen der Xlib, um Grafiken in die Zeichenfläche zu kopieren:

```
XImage *XCreateImage(Display *display,
                     Visual *visual,
                     unsigned int depth,
                     int format,
                     int offset,
                     char *data,
                     unsigned int width,
                     unsigned int height,
                     int bitmap_pad,
                     int bytes_per_line);

void XPutImage(Display *display,
               Drawable d,
```

```
                    GC gc,
                    XImage *image,
                    int src_x,
                    int src_y,
                    int dest_x,
                    int dest_y,
                    unsigned int width,
                    unsigned int height);

    void XDestroyImage(XImage *image);
```

Die Funktion `XCreateImage()` erzeugt eine Struktur vom Typ `XImage`, mit `XPutImage()` kann diese dann in ein „zeichenfähiges“ (*drawable*) X11-Objekt kopiert werden. Das X11-Window, das einer Zeichenfläche zu Grunde liegt, ist ein solches Objekt. Die für Xlib-Funktionen typische Vielzahl von Parametern wirkt zunächst etwas abschreckend, die Anwendung dieser Funktionen ist aber relativ unkompliziert – XView stellt Funktionen bereit, um die erforderlichen Objekte `display`, `visual` und `gc` zu erfragen. Mit der Funktion `XDestroyImage()` kann ein nicht mehr benötigtes XImage wieder „zerstört“ werden.
Das folgende Programm soll die Anwendung von `XCreateImage()`, `XPutImage()` und `XDestroyImage()` zum Darstellen einer Bitmap-Grafik in einer XView-Zeichnfläche demonstrieren. Es liest die als Parameter angegebene PPM-Grafikdatei (*Portable Pixmap*[10]) ein und zeigt diese in einer Zeichenfläche an:

```
 1  /*
 2      view_ppm.c - Portable Pixmap Grafik darstellen
 3  */
 4
 5  # include <stdio.h>
 6  # include <xview/frame.h>
 7  # include <xview/canvas.h>
 8  # include <xview/xv_xrect.h>
 9  # include <X11/Xutil.h>
10
11  # define KEY_IMAGE 1
12
13  void repaint_proc(Canvas canvas, Xv_Window paint_win,
14    Display *display, Window xwin, Xv_xrectlist *areas)
```

[10] Mit `pcxtoppm` lässt sich z.B. eine PCX-Grafik in das PPM-Format konvertieren.

```
15   {
16    GC gc;
17    XImage *image;
18
19    gc = DefaultGC(display, DefaultScreen(display));
20    image = (XImage *)xv_get(canvas,
21      XV_KEY_DATA, KEY_IMAGE);
22    XPutImage(display, xwin, gc, image, 0, 0, 0, 0,
23      image->width, image->height);
24    return;
25   }
26
27  int main(int argc, char *argv[])
28   {
29    Frame frame;
30    Display *display;
31    Visual *visual;
32    XImage *image;
33    int width, height;
34    FILE *stream;
35    char buffer[80], *data;
36
37    if ((argc != 2) || (strcmp(argv[1], "-h") == 0))
38     {
39      printf("Usage: view_ppm filename\n");
40      return(1);
41     }
42
43          /*- - - - - PPM-Datei einlesen - - - - -*/
44
45    if ((stream = fopen(argv[1], "r")) == NULL)
46     {
47      perror("view_ppm: Can't open file");
48      return(1);
49     }
50    if ((fgets(buffer, 80, stream) == NULL)
51        || (strcmp(buffer, "P6\n") != 0))
52     {
53      fprintf(stderr, "view_ppm: Bad PPM file.\n");
54      return(1);
55     }
```

```
56    do
57      if (fgets(buffer, 80, stream) == NULL)
58       {
59        fprintf(stderr, "view_ppm: Bad PPM file.\n");
60        return(1);
61       }
62    while(sscanf(buffer, "%d%d", &width, &height) != 2);
63    if ((fgets(buffer, 80, stream) == NULL)
64        || (strcmp(buffer, "255\n") != 0))
65     {
66      fprintf(stderr, "view_ppm: Unsupported format\n");
67      return(1);
68     }
69
70    if ((data = (void *)malloc(width*height*3)) == NULL)
71     {
72      fprintf(stderr, "view_ppm: Not enough memory.\n");
73      return(1);
74     }
75
76    fread(data, 3, width*height, stream);
77    fclose(stream);
78
79          /*- - - - - XView-Objekte erzeugen - - - - -*/
80
81    xv_init(XV_INIT_ARGC_PTR_ARGV, &argc, argv, 0);
82
83    frame = xv_create(0, FRAME,
84      XV_WIDTH,                 width+2,
85      XV_HEIGHT,                height+2,
86      FRAME_SHOW_RESIZE_CORNER, FALSE,
87      FRAME_LABEL,              argv[1],
88      0);
89
90    display = (Display *)xv_get(frame, XV_DISPLAY);
91    if (DefaultDepth(display, DefaultScreen(display))
92      != 24)
93     {
94      fprintf(stderr, "view_ppm: Set X11 to 24 bpp.\n");
95      return(1);
96     }
```

```
 97    visual = (Visual *)xv_get(frame, XV_VISUAL);
 98
 99    image = XCreateImage(display, visual, 24, ZPixmap,
100      0, data, width, height, 16, width*3);
101    image->byte_order = MSBFirst;
102
103    xv_create(frame, CANVAS,
104      CANVAS_AUTO_SHRINK,    FALSE,
105      CANVAS_AUTO_EXPAND,    FALSE,
106      CANVAS_REPAINT_PROC,   repaint_proc,
107      CANVAS_X_PAINT_WINDOW, TRUE,
108      XV_KEY_DATA,           KEY_IMAGE, image,
109      0);
110
111    xv_main_loop(frame);
112    XDestroyImage(image);
113    return(0);
114   }
```

Die Callback-Funktion `repaint_proc()` (Zeile 13 bis 25) kopiert die in die `XImage`-Struktur eingebettete Grafik mit Hilfe von `XPutImage()` in die Zeichenfläche. Im Hauptprogramm ab Zeile 27 werden zunächst die Kommandozeilenparameter ausgewertet und die angegebene Datei geöffnet. In den Zeilen 45 bis 68 erfolgt das Einlesen des Dateikopfes mit den Informationen über Breite und Höhe der Grafik. Mit diesen Daten kann dann der Speicher für die Grafik angefordert werden, um anschließend die eigentliche Bildinformation aus der Datei in diesen Speicher einzulesen (Zeile 76). Danach wird das Fenster erzeugt, wobei Breite und Höhe um zwei Pixel größer als das Bild gewählt werden. Dadurch wird die Umrandung der Zeichenfläche (an allen vier Seiten ein Pixel breit) berücksichtigt. In Zeile 91 und 92 wird überprüft, ob der X-Server mit einer Farbtiefe von 24 Bit pro Pixel läuft, denn beim Kopieren eines XImages in ein X11-Window müssen beide die gleiche Farbtiefe besitzen. Ist diese Bedingung erfüllt, wird das XImage in Zeile 99 und 100 erzeugt und anschließend die gleiche Reihenfolge der Bytes (höchstes Byte zuerst) wie in der PPM-Datei eingestellt (Zeile 101). Im letzten Schritt wird dann die Zeichenfläche erzeugt und das XImage als `XV_KEY_DATA` angehängt.

Durch den Aufruf der Funktion `XDestroyImage()` wird nicht nur der Speicher für die XImage-Struktur wieder freigegeben, sondern auch der Bildspeicher (hier: `data`). Ein zusätzlicher Aufruf der Funktion `free()` würde zu einer Speicherzugriffsverletzung führen.

Wenn Sie das gleiche Programm bei einer anderen Farbtiefe des X-Servers als 24 Bit verwenden wollen, müssen die Bilddaten zuerst umgerechnet werden. Bei 256 Farben oder weniger verwendet der X-Server ein ganz anderes Farbmodell, sodass ein einfaches Umrechnen nicht mehr möglich ist.

Eine andere Möglichkeit, Bitmap-Grafiken in einer Zeichenfläche darzustellen, liefert die Funktionsbibliothek `libXpm`. Diese beinhaltet Funktionen speziell zur Handhabung von X11-Pixmap-Dateien (XPM). Dabei übernimmt die Bibliothek auch das Laden (und Speichern) solcher Dateien. Das folgende Programm „`view_xpm`“ läd die als Parameter angegebene X11-Pixmap-Datei und stellt die Grafik in einer Zeichenfläche dar.

```
1  /*
2      view_xpm.c - X11-Pixmap darstellen
3  */
4
5  # include <stdio.h>
6  # include <xview/frame.h>
7  # include <xview/canvas.h>
8  # include <xview/xv_xrect.h>
9  # include <X11/xpm.h>
10
11 # define KEY_PIXMAP 1
12 # define KEY_WIDTH 2
13 # define KEY_HEIGHT 3
14
15 void repaint_proc(Canvas canvas, Xv_Window paint_win,
16   Display *display, Window xwin, Xv_xrectlist *areas)
17  {
18   GC gc;
19   Pixmap pixmap;
20   int width, height;
21
22   gc = DefaultGC(display, DefaultScreen(display));
23   pixmap = xv_get(canvas, XV_KEY_DATA, KEY_PIXMAP);
24   width = xv_get(canvas, XV_KEY_DATA, KEY_WIDTH);
25   height = xv_get(canvas, XV_KEY_DATA, KEY_HEIGHT);
26   XCopyArea(display, pixmap, xwin, gc, 0, 0,
27     width, height, 0, 0);
28   return;
```

```
29   }
30
31  int main(int argc, char *argv[])
32   {
33    int width, height, error;
34    Frame frame;
35    Canvas canvas;
36    Window xwin;
37    Display *display;
38    Pixmap pixmap, shape;
39    XpmAttributes attributes;
40
41    if ((argc != 2) || (strcmp(argv[1], "-h") == 0))
42     {
43      printf("Usage: view_xpm filename\n");
44      return(1);
45     }
46
47          /*- - - - - XView-Objekte erzeugen - - - - -*/
48
49    xv_init(XV_INIT_ARGC_PTR_ARGV, &argc, argv, 0);
50
51    frame = xv_create(0, FRAME,
52      FRAME_LABEL,              argv[1],
53      FRAME_SHOW_RESIZE_CORNER, FALSE,
54      0);
55
56    display = (Display *)xv_get(frame, XV_DISPLAY);
57    xwin = xv_get(frame, XV_XID);
58
59    attributes.valuemask = 0;
60    error = XpmReadFileToPixmap(display, xwin, argv[1],
61      &pixmap, &shape, &attributes);
62    if (error)            /* nicht genug Farben frei? */
63     {
64      fprintf(stderr, "view_xpm: Can't read pixmap.\n");
65      return(1);
66     }
67
68    width = attributes.width;
69    height = attributes.height;
```

```
70
71    xv_set(frame,
72      XV_WIDTH,  width+2,
73      XV_HEIGHT, height+2,
74      0);
75
76    canvas = xv_create(frame, CANVAS,
77      CANVAS_REPAINT_PROC,   repaint_proc,
78      CANVAS_X_PAINT_WINDOW, TRUE,
79      XV_KEY_DATA,           KEY_PIXMAP, pixmap,
80      XV_KEY_DATA,           KEY_WIDTH, width,
81      XV_KEY_DATA,           KEY_HEIGHT, height,
82      0);
83
84    xv_main_loop(frame);
85    XFreePixmap(display, pixmap);
86    return(0);
87   }
```

Beim Übersetzen des Programms muss die Bibliothek `libXpm` eingebunden werden, der vollständige Compiler-Aufruf lautet damit:

```
gcc -I/usr/openwin/include -L/usr/X11/lib
-L/usr/openwin/lib view_xpm.c -lX11 -lXpm -lxview
-lolgx -o view_xpm
```

Mit der Funktion `XpmReadFileToPixmap()` in Zeile 60 wird die angegebene XPM-Datei geladen und ein Pixmap-Objekt erzeugt. Ferner wird die Farbpalette der Grafik angepasst. Die Eigenschaften der Grafik wie Breite und Höhe werden in der Struktur `XpmAttributes` eingetragen. Mit der Xlib-Funktion `XCopyArea()` (Zeile 26) kann die so erzeugte Pixmap (oder ein Ausschnitt davon) an eine beliebige Stelle in der Zeichenfläche kopiert werden.
Im Gegensatz zu dem vorherigen Beispielprogramm `view_ppm` ist hier keine Überprüfung der Farbtiefe des X-Servers notwendig. Die `libXpm` übernimmt das Umrechnen der Grafik auf die aktuelle Farbenzahl. Können jedoch nicht genügend Farben reserviert werden, so erzeugt `XpmReadFileToPixmap()` keine Pixmap und liefert einen Rückgabewert ungleich 0.
Hinweis: Pixmap-Dateien bieten die Möglichkeit einer Schablone, die angibt, welche Bereiche der Grafik durchsichtig sind. Das Beispielprogramm unterstützt diese Funktion jedoch nicht.

Bild 8.16: Der Linux-Pinguin als XPM-Datei – dargestellt mit `view_xpm`

8.2.17 Ein kleines Zeichenprogramm

In den bisherigen Beispielen wurde die Zeichenfläche (Canvas) nur zur Darstellung von Grafik verwendet. Eine solche Zeichenfläche kann aber auch als Eingabeobjekt – z.B. für Tastatureingaben und Mausklicks – verwendet werden. Dies soll anhand des folgenden Quelltextes für ein kleines Zeichenprogramm[11] demonstriert werden. Bei diesem Programm kann über die Tastatur die aktuelle Zeichenfunktion gewählt werden: „l“=Linie, „r“=Rechteck, „e“=Ellipse und Leertaste zum Löschen der Zeichenfläche, wobei die eingestellte Zeichenfunktion in der Fußzeile des Fensters dargestellt ist (siehe Bild 8.17). Mit der Maus lassen sich dann Linien, Rechtecke und Ellipsen zeichnen; mit der Taste „q“ wird das Programm beendet.

```
1  /*
2      xvdraw.c - Linien, Ellipsen und Rechtecke
3                 zeichnen
4  */
5
6  # include <xview/frame.h>
7  # include <xview/canvas.h>
8  # include <xview/xv_xrect.h>
9
10 # define KEY_DRAWING 1
11 # define KEY_FRAME 2
12 # define MAX_OBJECTS 100
13
14 typedef struct
15  {
16   enum {line, ellipse, rect} type;
```

[11] Für ein „richtiges“ Zeichenprogramm fehlt natürlich noch eine Funktion zum Speichern der Grafik.

```
17    int x1, y1, x2, y2;
18   } object;
19
20  typedef struct
21   {
22    int num_objects;
23    object object[MAX_OBJECTS];
24   } drawing;
25
26  void draw_object(Display *display, Window xwin,
27    GC gc, object *obj)
28   {
29    int x1, y1, x2, y2, x0, y0, dx, dy;
30
31    x1 = obj->x1;
32    y1 = obj->y1;
33    x2 = obj->x2;
34    y2 = obj->y2;
35    if ((x1 == x2) && (y1 == y2))
36      return;
37
38    if (obj->type == line)
39      XDrawLine(display, xwin, gc, x1, y1, x2, y2);
40    else
41     {
42      x0 = MIN(x1, x2);
43      y0 = MIN(y1, y2);
44      dx = abs(x2-x1);
45      dy = abs(y2-y1);
46      if (obj->type == ellipse)
47        XDrawArc(display, xwin, gc, x0, y0, dx, dy,
48        0, 64*360);
49      else
50        XDrawRectangle(display, xwin, gc, x0, y0,
51          dx, dy);
52     }
53    return;
54   }
55
56  void repaint_proc(Canvas canvas, Xv_Window paint_win,
57    Display *display, Window xwin, Xv_xrectlist *areas)
```

```
58  {
59   int i;
60   drawing *drwg;
61   GC gc;
62
63   gc = DefaultGC(display, DefaultScreen(display));
64   drwg = (drawing *)xv_get(paint_win,
65          XV_KEY_DATA, KEY_DRAWING);
66   for (i=0; i<drwg->num_objects; i++)
67    draw_object(display, xwin, gc, &(drwg->object[i]));
68   return;
69  }
70
71 void event_proc(Xv_Window win, Event *event)
72  {
73   int n;
74   drawing *drwg;
75   object *obj;
76   Frame frame;
77   Display *display;
78   Window xwin;
79   GC gc;
80
81   display = (Display *)xv_get(win, XV_DISPLAY);
82   xwin = xv_get(win, XV_XID);
83   frame = xv_get(win, XV_KEY_DATA, KEY_FRAME);
84   drwg = (drawing *)xv_get(win,
85               XV_KEY_DATA, KEY_DRAWING);
86   n = drwg->num_objects;
87   obj = &(drwg->object[n]);
88
89   if (n == MAX_OBJECTS)
90    {
91     xv_set(frame, FRAME_LEFT_FOOTER,
92       "maximale Anzahl von Objekten erreicht!",
93       0);
94     if (event_is_ascii(event)
95         && (event_action(event) == 'q'))
96       xv_destroy_safe(frame);
97     return;
98    }
```

```
 99
100     if (event_is_ascii(event))
101      {
102       switch(event_action(event))
103        {
104         case 'q': xv_destroy_safe(frame);
105           return;
106         case 'l': obj->type = line;
107           xv_set(frame, FRAME_LEFT_FOOTER, "Linie", 0);
108           break;
109         case 'e': obj->type = ellipse;
110           xv_set(frame, FRAME_LEFT_FOOTER,"Ellipse", 0);
111           break;
112         case 'r': obj->type = rect;
113           xv_set(frame, FRAME_LEFT_FOOTER,"Rechteck",0);
114           break;
115         case ' ': drwg->num_objects = 0;
116           drwg->object[0].type = obj->type;
117           XClearWindow(display, xwin);
118           break;
119        }
120       obj->x1 = obj->x2;
121       obj->y1 = obj->y2;
122      }
123     else  /* Maustaste oder -bewegung */
124      {
125       gc = DefaultGC(display, DefaultScreen(display));
126       if (event_action(event) == ACTION_SELECT)
127        {
128         if (event_is_down(event))
129          {
130           obj->x1 = event_x(event);
131           obj->x2 = event_x(event);
132           obj->y1 = event_y(event);
133           obj->y2 = event_y(event);
134          }
135         else
136          {
137           XSetFunction(display, gc, GXcopy);
138           draw_object(display, xwin, gc, obj);
139           if (n < MAX_OBJECTS-1)
```

```
140           drwg->object[n+1].type = obj->type;
141          drwg->num_objects++;
142         }
143       }
144      else if (event_action(event) == LOC_DRAG)
145       {
146        XSetFunction(display, gc, GXinvert);
147        draw_object(display, xwin, gc, obj);
148        obj->x2 = event_x(event);
149        obj->y2 = event_y(event);
150        draw_object(display, xwin, gc, obj);
151       }
152     }
153    return;
154   }
155
156  /*- - - - - - - - - - - - - - - - - - - - - - - - -*/
157
158  int main(int argc, char *argv[])
159   {
160    drawing drwg;
161    Frame frame;
162    Canvas canvas;
163
164    drwg.num_objects = 0;
165    drwg.object[0].type = line;
166
167    xv_init(XV_INIT_ARGC_PTR_ARGV, &argc, argv, 0);
168
169    frame = xv_create(0, FRAME,
170      FRAME_LABEL,       argv[0],
171      XV_WIDTH,          400,
172      XV_HEIGHT,         300,
173      FRAME_SHOW_FOOTER, TRUE,
174      FRAME_LEFT_FOOTER, "Linie",
175      0);
176
177    canvas = xv_create(frame, CANVAS,
178      CANVAS_REPAINT_PROC,   repaint_proc,
179      CANVAS_X_PAINT_WINDOW, TRUE,
180      0);
```

```
181
182     xv_set(canvas_paint_window(canvas),
183       WIN_EVENT_PROC,      event_proc,
184       WIN_CONSUME_EVENTS,
185         WIN_MOUSE_BUTTONS,
186         LOC_DRAG,
187         WIN_ASCII_EVENTS,
188         0,
189       XV_KEY_DATA,         KEY_DRAWING, &drwg,
190       XV_KEY_DATA,         KEY_FRAME, frame,
191       0);
192
193     xv_main_loop(frame);
194     return(0);
195   }
```

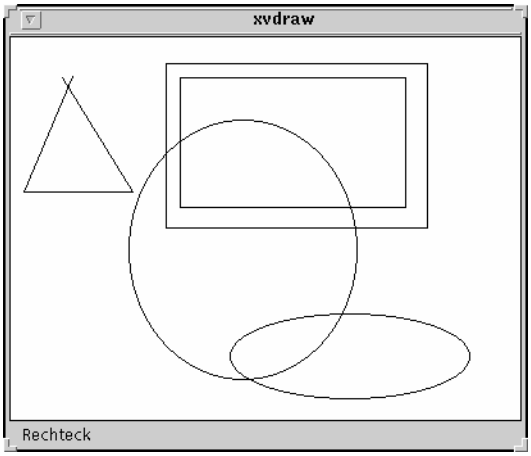

Bild 8.17: Ein kleines Zeichenprogramm

Betrachten wir zunächst das Hauptprogramm ab Zeile 158. Nach dem Erzeugen des Fensters und der Zeichenfläche wird in den Zeilen 182 bis 191 die Funktion zur Bearbeitung von Ereignissen (*Events*), die innerhalb der Zeichenfläche auftreten, eingestellt. Mit dem Attribut `WIN_CONSUME_EVENTS` wird festgelegt, *welche* Ereignisse registriert werden, wobei `LOC_DRAG` für das Erfassen von Mausbewegungen mit gedrückter Maustaste steht. Mit `LOC_MOVE` würden Mausbewegungen ohne ge-

drückte Maustaste erfasst. Die Funktion `event_proc()` zur „Ereignisbehandlung“ ist in den Zeilen 71 bis 154 definiert. Die eigentliche Auswertung des Ereignisses erfolgt ab Zeile 100. Mit `event_is_ascii()` kann abgefragt werden, ob es sich bei dem Ereignis um eine Tastatureingabe handelt. In diesem Fall liefert `event_action()` das ASCII-Zeichen der gedrückten Taste. Wurde stattdessen die linke Maustaste gedrückt, liefert `event_action()` den Wert `ACTION_SELECT` (Zeile 126), die rechte Maustaste erzeugt dagegen `ACTION_MENU`. Auch das Loslassen der Maustasten erzeugt ein solches Ereignis, über `event_is_down()` kann jedoch erfragt werden, ob die Taste gedrückt oder losgelassen wurde (Zeile 128).
Während man die Maus mit gedrückter Maustaste zum zweiten Eckpunkt einer Linie, Ellipse oder eines Rechtecks bewegt, wird das zu zeichnende Objekt immer wieder entsprechend der aktuellen Mausposition neu dargestellt. Um hierbei nicht die bereits gezeichneten Objekte zu zerstören, werden die Bildpunkte des neuen Objektes nicht einfach gesetzt. Stattdessen wird dort die bereits bestehende Grafik invertiert. Durch nochmaliges Invertieren lässt sich so der ursprüngliche Zustand wiederherstellen. Erreicht wird dieser Zeichenmodus durch die Wahl der Verknüpfungsfunktion `GXinvert` mit `XSetFunction()` in Zeile 146. Wird die Maustaste losgelassen – der zweite Eckpunkt des Objektes steht damit fest –, stellt der `XSetFunction()`-Aufruf in Zeile 137 die Verknüpfungsfunktion wieder auf `GXcopy`, wodurch die Vordergrundfarbe auf die Bildpunkte des neuen Objektes kopiert wird. Die Xlib bietet neben `GXcopy` und `GXinvert` noch eine ganze Reihe weiterer Verknüpfungsfunktionen, die mit

```
grep GX /usr/include/X11/X.h
```

aufgelistet werden können.

8.2.18 Dateiauswahlfenster

Neben den bisher vorgestellten Objekten bietet XView noch ein paar weitere, zum Teil relativ komplexe Objekte wie beispielsweise Texteditor- oder Terminalfenster. Komplex bedeutet in diesem Zusammenhang nicht kompliziert in der Bedienung, sondern lediglich, dass diese Objekte aus mehreren einfachen Objekte zusammengesetzt sind. Eines dieser komplexeren Objekte ist das Dateiauswahlfenster (*File Chooser*) zum Laden und Speichern von Dateien. Das folgende Programm öffnet bei Betätigung der Schaltfläche „Datei laden“ ein solches Dateiauswahlfenster (siehe Bild 8.18).

```
1  /*
2      choose_file.c - Dateiauswahlfenster oeffnen
3  */
4
5  # include <stdio.h>
6  # include <unistd.h>
7  # include <xview/frame.h>
8  # include <xview/panel.h>
9  # include <xview/file_chsr.h>
10
11 void quit_proc(Panel_item item, Event *event)
12  {
13   xv_destroy_safe(xv_get(item, PANEL_CLIENT_DATA));
14   return;
15  }
16
17 void choose_file_proc(Panel_item item, Event *event)
18  {
19   File_chooser file_chooser;
20
21   file_chooser = xv_get(item, PANEL_CLIENT_DATA);
22   xv_set(file_chooser, XV_SHOW, TRUE, 0);
23   return;
24  }
25
26 int load_proc(File_chooser file_ch, char *filename)
27  {
28   printf("Dateiname: '%s'\n", filename);
29   return(XV_OK);
30  }
31
32 /*- - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
33
34 int main(int argc, char *argv[])
35  {
36   Frame frame;
37   Panel panel;
38   File_chooser file_chooser;
39
40   xv_init(XV_INIT_ARGC_PTR_ARGV, &argc, argv,
41     XV_USE_LOCALE, TRUE, 0);
```

```
42
43    frame = xv_create(0, FRAME,
44      FRAME_LABEL,      argv[0],
45      XV_WIDTH,         200,
46      XV_HEIGHT,        100,
47      0);
48
49    file_chooser = xv_create(frame, FILE_CHOOSER,
50      FILE_CHOOSER_TYPE,        FILE_CHOOSER_OPEN,
51      FILE_CHOOSER_NOTIFY_FUNC, load_proc,
52      0);
53
54    panel = xv_create(frame, PANEL, 0);
55
56    xv_create(panel, PANEL_BUTTON,
57      PANEL_LABEL_STRING, "Datei laden",
58      PANEL_NOTIFY_PROC,  choose_file_proc,
59      PANEL_CLIENT_DATA,  file_chooser,
60      0);
61
62    xv_create(panel, PANEL_BUTTON,
63      PANEL_LABEL_STRING, "Ende",
64      PANEL_NOTIFY_PROC,  quit_proc,
65      PANEL_CLIENT_DATA,  frame,
66      0);
67
68    xv_main_loop(frame);
69    return(0);
70  }
```

Sollte bei Ihnen das Dateiauswahlfenster mit englischen Texten erscheinen, lesen Sie bitte Anhang A3. Im Allgemeinen sollte das Attribut `XV_USE_LOCALE` in Zeile 41 zur Anpassung an die eingestellte Landessprache führen.
Betrachten wir zunächst das Hauptprogramm ab Zeile 34. Nach Erzeugen des Hauptfensters wird bereits das Dateiauswahlfenster erzeugt (Zeilen 49 bis 52). Ähnlich wie ein *Command Frame* (siehe oben) wird auch das Dateiauswahlfenster nicht sofort sichtbar – es erscheint erst durch Setzen des Attributs `XV_SHOW` auf `TRUE`. Dadurch braucht das Dateiauswahlfenster nicht für jedes Öffnen oder Speichern einer Datei neu erzeugt zu werden, sondern muss lediglich „sichtbar“ gemacht werden.

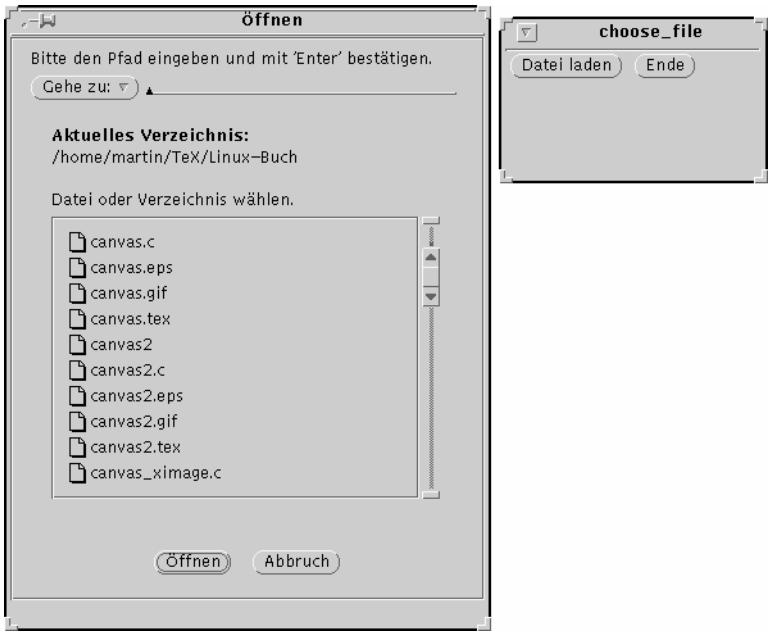

Bild 8.18: Dateiauswahlfenster der XView-Bibliothek

Mit dem Attribut `FILE_CHOOSER_TYPE` kann zwischen einem Fenster zum Laden (`OPEN`) und Speichern (`SAVE` oder `SAVEAS`) gewählt werden. `FILE_CHOOSER_NOTIFY_FUNC` spezifiziert die Funktion, die beim Klicken auf die Schaltfläche „Öffnen“ aufgerufen wird. Weitere mögliche Attribute sind:

`XV_LABEL` Angabe des Textes in der Kopfzeile des Auswahlfensters.

`FILE_CHOOSER_DOC_NAME` Voreinstellung für den Dateinamen (nur beim Speichern).

`FILE_CHOOSER_DIRECTORY` Angabe des Startverzeichnisses.

`FILE_CHOOSER_SHOW_DOT_FILES` Versteckte Dateien anzeigen (`TRUE`) oder nicht (`FALSE`).

`FILE_CHOOSER_FILTER_STRING` Angabe eines „regulären Ausdrucks“ (*regular expression*[12]), mit dem vorgegeben werden kann, welche

[12] Die Syntax solcher Ausdrücke ist z.B. in der „man“-Seite von `regexp` beschrieben.

Dateinamen selektierbar sind. Beispiel: `".*[.][ch]$"` stellt nur Dateien dar, die auf „`.c`“ oder „`.h`“ enden.

Nach dem Dateiauswahlfenster werden in den Zeilen 54 bis 66 noch das Bedienfeld und zwei Schaltflächen für das Hauptfenster erzeugt – die eine zum Öffnen des Dateiauswahlfensters und die andere zum Beenden des Programms. In der mit der Schaltfläche „Datei laden“ verbundenen Callback-Funktion `choose_file_proc()` wird lediglich das Dateiauswahlfenster mit dem Attribut `XV_SHOW` sichtbar gemacht (Zeile 22). Bei Betätigung der Schaltfläche „Öffnen“ innerhalb des Dateiauswahlfensters wird die Funktion `load_proc()` aufgerufen, da diese Funktion beim Erzeugen des Auswahlfensters als `NOTIFY_FUNC` eingetragen wurde. Der Funktion werden zwei Parameter übergeben: das Objekt des Auswahlfensters und der selektierte Dateiname. Gibt die Funktion den Wert `XV_OK` zurück, wird das Auswahlfenster automatisch geschlossen. Bei `XV_ERROR` würde das Fenster geöffnet bleiben und dem Anwender so die Möglichkeit geben, eine andere Datei zu selektieren.

Ein (kleiner) XView-Bug

Leider hat die XView-Bibliothek, Version 3.2, einen kleinen Bug: beim Auswerten der X11-Ressourcen[13] kann es zu einem Speicherüberlauf kommen, der XView-interne Variablen zerstört. Dies kann beispielsweise beim Beenden des Programms zu folgender Fehlermeldung führen:

```
XView warning: Notifier error: Unknown client
```

Als Download zum Buch ist eine überarbeitete Version der `libxview` verfügbar, mit der dieses Problem nicht mehr auftritt.

[13] Das sind Einstellungen für die grafische Oberfläche und die X11-Programme. Mit „`xrdb -query`“ können Sie sich diese Einstellungen ansehen.