1 Dialogs

- Windows used to *give information* to the user or *accept inputs* from the user
- Can be *predefined* or *customised*
- There are two types of dialogs *Modal* and *Modeless* (this is usually applicable to other windowing systems also)

2 Predefined Dialogs

2.1 Modal Dialog

- *Must* be answered before anything else is done
- Other operations cannot be performed until the dialog has been dealt with
- Other windows or pull down menus cannot be selected whilst a modal dialog is displayed
- Cannot be moved from their position of display

2.2 Modeless Dialog

- Does not need to be answered *immediately*
- Can select pull down menu commands, other windows or run other Prolog queries before responding to the dialog
- If another Prolog query is selected - the process that created the Modeless dialog will be suspended
- Can be moved around the screen with a mouse

3 Dialog categories

3.1 Obtaining user input

```
prompt_read(Prompt_list, Term)
ask(Prompt_list,Input)
```

- Both are *modeless dialogs* - with an edit field into which the required term is to be entered
- *Prompt_list* is the sentence displayed within the dialog, and *Term* is the variable to which the input of the user is bound
- If the user simply presses return - or clicks on the *OK* button, *Term* is bound to the built in *end_of_file*.

```
read1 :-
  prompt_read(['Enter a likes clause'],Clause).

read2(Name, Feeling) :-
  ask(['How are you,', Name, ?],Feeling),
  write(Feeling).
```

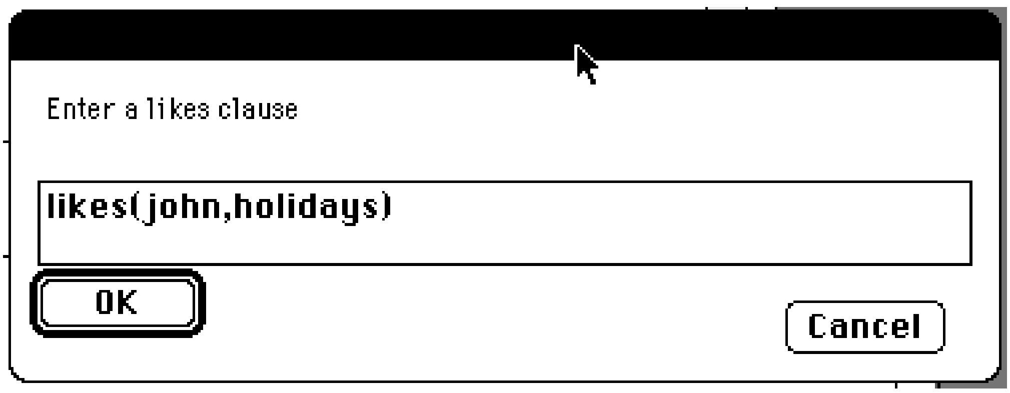

Figure 1: Prompt dialog

- For the *ask* dialog, no syntax checks are made on the input

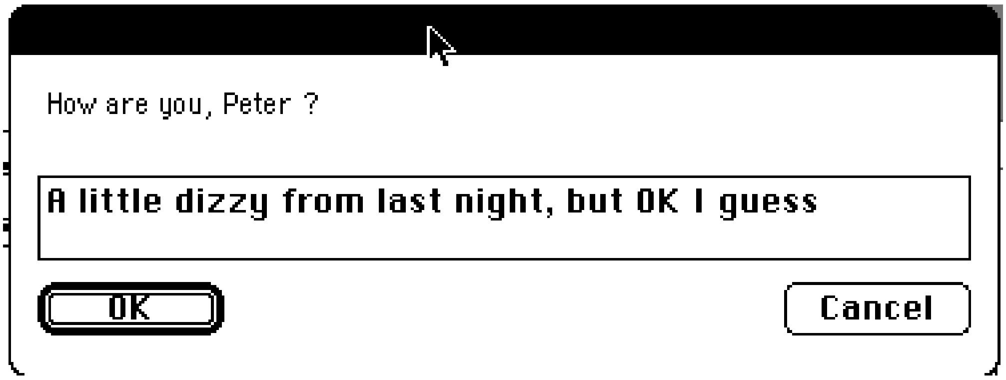

Figure 2: Ask dialog

- The *Input* argument - in the example this is the variable *Feeling* will be bound to a list of tokens entered in the edit field, hence :

```
Feeling =
 ['A', little, dizzy, from, last, night, ',',
     but,'OK','I',guess]
```

- This can then be processed by the calling program (as required).

```
menu(Music) :-
 scroll_menu(['What music do you like ?'],
 ['Pop','Indie','Jazz','Metal','Punk',
 'Fusion','Classical','Reggae','Dance'],
 ['Jazz'], Music).
```

- Displays a scrolling menu for selecting *one or more* items
- The menu dialog is *modal*
- The general form is :

```
scroll_menu(Prompt,Menu_list,Preselected,Selected)
```

- *Prompt* refers to the question asked from the user, *Menu_list* corresponds to the different choices offered, *Preselected* is the item highlighted when the menu is invoked and *Selected* is a variable containing the choices made by the user

- In the above example, if the user selected *Jazz*, then

```
Music = ['Jazz']
```

- If multiple options were selected (by holding down the shift key and pressing the mouse button), then

```
Music = ['Metal','Punk','Dance']
```

- If the atom 'Jazz' instead of the list

```
['Jazz']
```

  was the *Preselected* argument in the example, then only ONE option would be selectable from the menu list

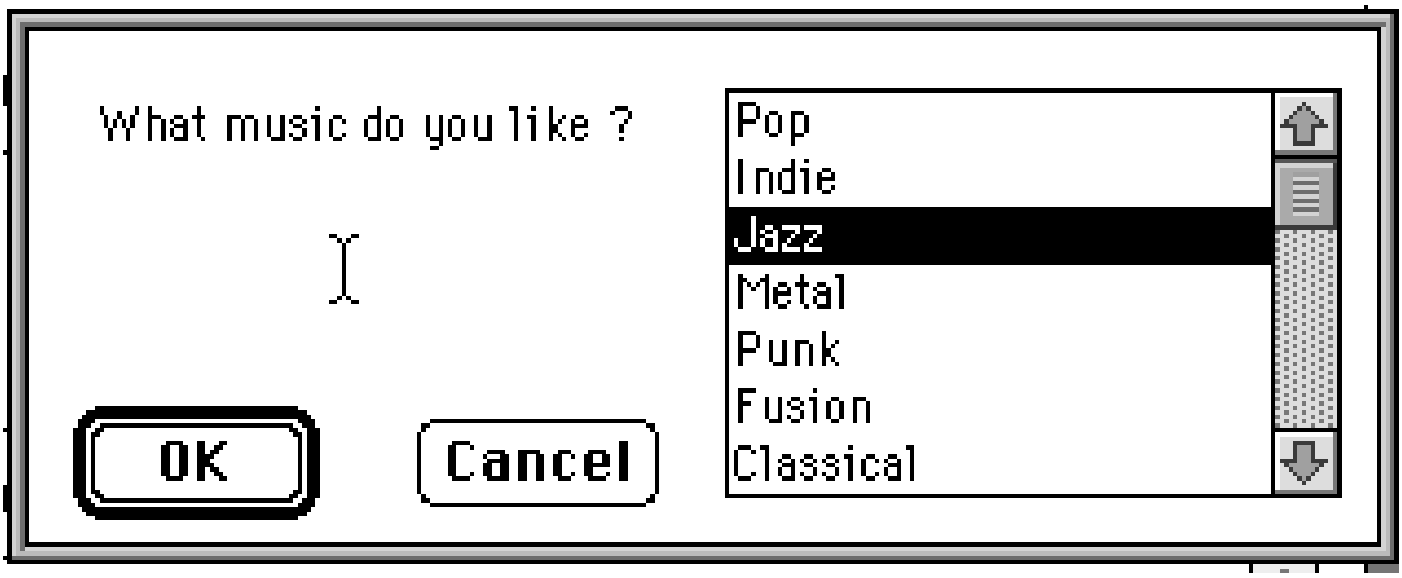

Figure 3: Dialog Menu

- A simple dialog can be used to obtain a *yes/no* type of response from the user. This is achieved by the use of the modal and modeless yes/no dialog

  ```
  yesno(Prompt_list)
  myesno(Prompt_list)
  ```

- *Prompt_list* is a sequence of terms separated by spaces

```
question1 :-
 yesno(['Is the moon green',?]).

question2 :-
  myesno(['Have you switched off the gas',?]).
```

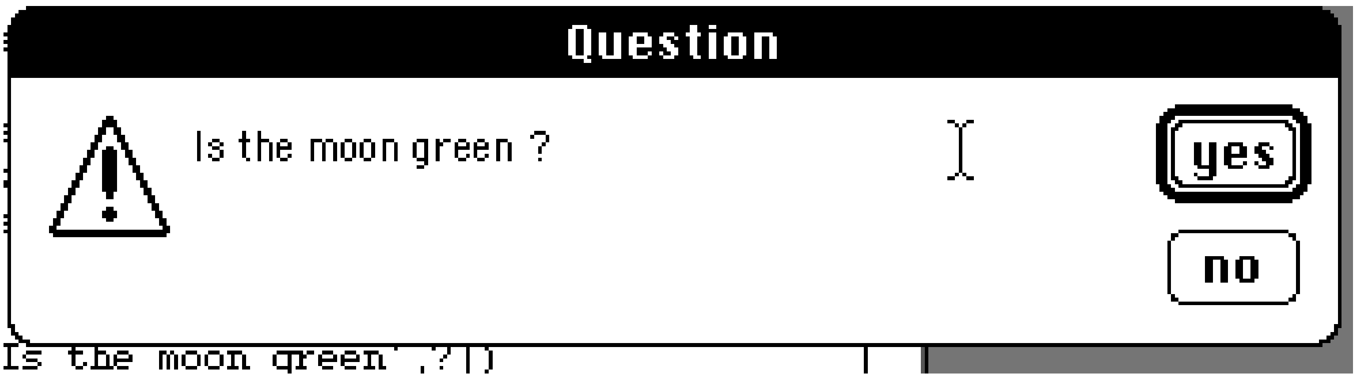

Figure 4: The *Modeless* yes/no dialog

3.2 Messages to the user

- There are four ways to give messages to the user - i.e. where no user input is required directly
- These predefined dialogs are :

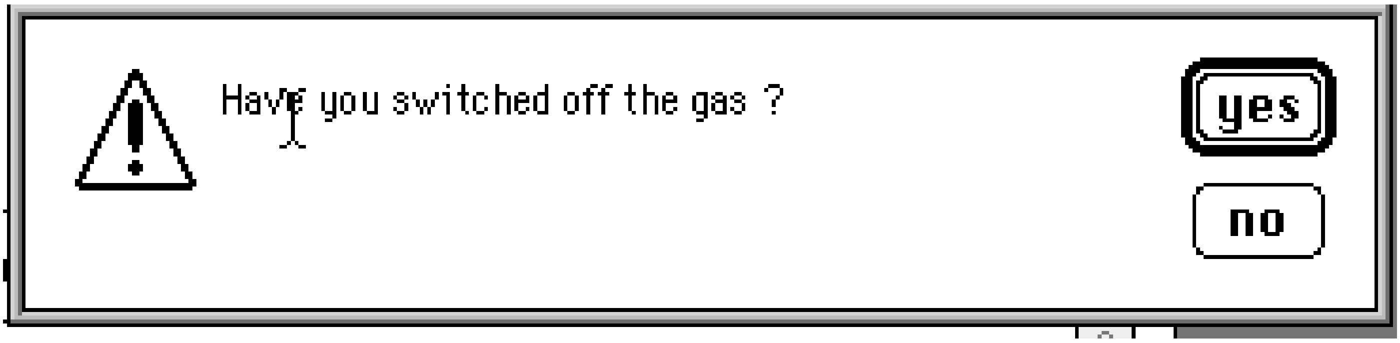

Figure 5: The *Modal* yes/no dialog

```
message(Term_list)
warning(Term_list)
errormessage(Term_list)
banner(Call,Message,T,L)
```

- The *Term_list* corresponds a list of items (message) to be displayed
- *Call* in the *banner* case refers to a *goal* or *predicate* to be executed - and *Message* is the message to be displayed to the user - *T* and *L* correspond to the top and left coordinates of the banner window
- Note that all of these dialogs are *Modal*

```
availability(Album) :-
 album(Album),
 message([Album,' can be found in section 2 of the store.']

album(sketches_of_spain).
album(aura).

warn :-
 warning(['Uncle Sam is looking at you!']).
```

```
error :-
  errormessage(['Uncle Sam wants you dead']).

longwait(0).
longwait(X) :-
 X1 is X-1,
 longwait(X1).

flag :-
 banner(longwait(1000),['Please wait...'],50,10).
```

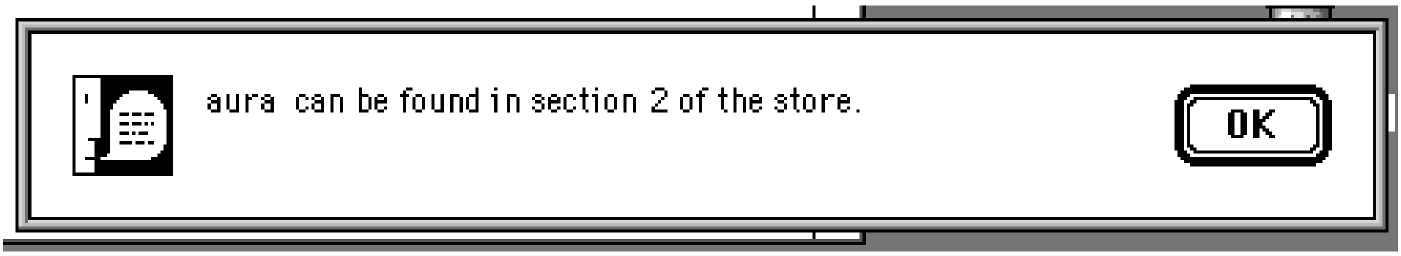

Figure 6: The message dialog

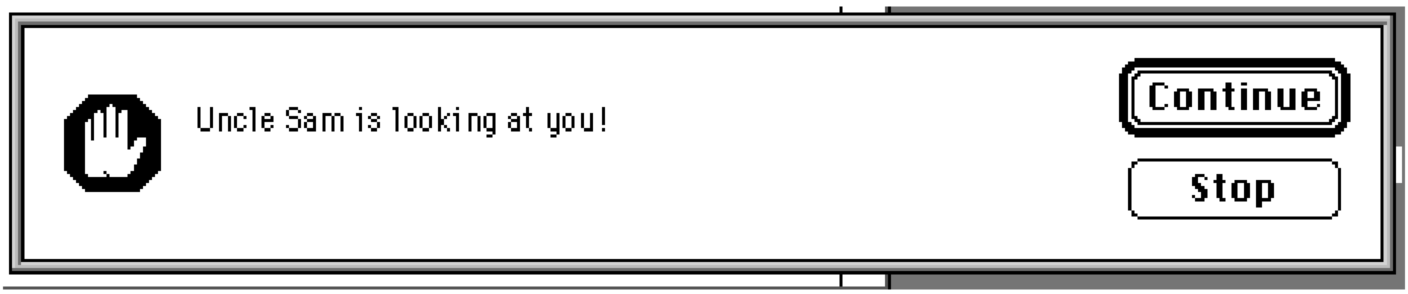

Figure 7: The *Modal* warning dialog

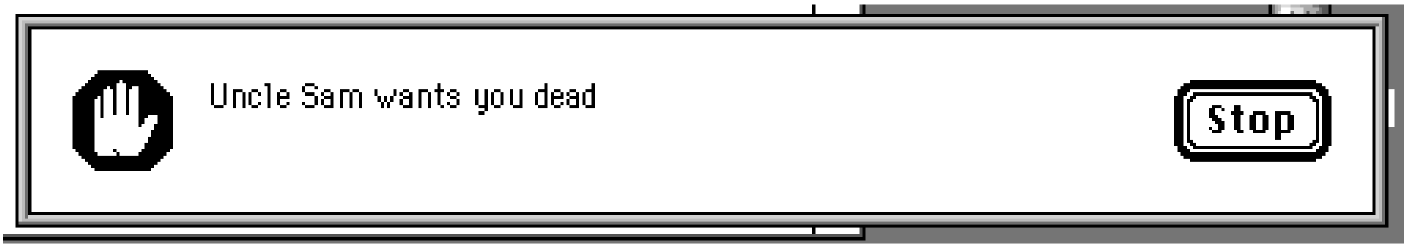

Figure 8: The *Modal* error message dialog

Figure 9: The *Modal* banner dialog

4 Custom Dialogs

Custom dialogs are composed of *buttons, icons, text fields, edit fields, check boxes, radio buttons and scrolling menus*

- Useful, because it is possible to perform checks on what the user inputs (such as type or range checks)
- Also gives the designer more power to make the dialog more expressive - hence greater control over the user interaction

```
dialog(Title, Top, Left, Depth, Width,
   DItems, Response)
```

- *Title* corresponds to the title displayed on the dialog
- *Top, Left, Depth and Width* correspond to position the dialog on the screen
- *DItems* correspond to a list of choices to be made by the user - and are generally referred to as *dialog field format descriptors*

- *Response* is a variable which indicates the selected button

```
start :-
 dialog('Destruct',60,20,70,160,
 [button( 37,87,26,66,'Ok'),
  button( 40,10,20,60,'Cancel'),
  text(10,12,20,140,'Launch Missile')],
  Response).
```

This will lead to a dialog with the title *Destruct* placed at the part of the screen specified by the first 4 coordinates - it also has 2 *buttons* and a *text* field above the buttons.

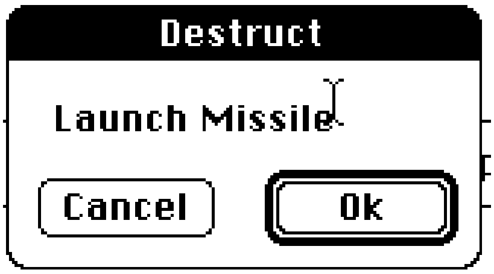

Figure 10: The *Missle* box

- If user presses *Ok* - call succeeds and *Response* is bound to 1
- If user presses *Cancel* - call fails
- Normally - first two items *should be buttons* (as these are handled in a special way)

4.1 Field Description

Each field within the dialog has the standard form :

```
Field_type(Top, Left, Depth, Width, ...)

eg. button( 37,87,26,66,'Ok')
or text(10,12,20,140,'Launch Missile')
```

- The *Top* and *Left* arguments in this definition and the dialog definition are in *pixels* and are relative to the *dialog* window NOT the complete screen

4.2 Dialog primitives

Buttons

```
button(Top,Left,Depth,Width,Label)

check(Top,Left, Depth, Width, Label,
       Initial Value, Final Value)

radio(Top,Left,Depth,Width,Label,
       Initial Value, Final Value)

text(Top,Left,Depth,Width,
   Text Value)

scrolltext(Top, Left,Depth,Width, Text)

scrolltext(Top, Left,Depth,Width,
  Text, Font, Size, Style)
```

- *check* and *radio* correspond to check buttons and radio button respectively - which can be either *ON* or *OFF*

- The *Initial Value* correspond to one of these states - whereas the *Final Value* is a variable which takes on a value set by the user - and bound to either *ON* or *OFF*
- All *buttons*, *radio buttons* or *check buttons* should be put adjacent to each other as a group
- Only **one** *radio button* can be chosen - BUT can have multiple *check buttons*
- Multiple *versions* of *text* and *scrolltext* exist - depending on whether *Font*, *Style* and *Size* of text is specified - it is also possible to set text *Colour*

```
carpanel :-
 dialog('Test Car',50,150,210,310,
 [button(142,157,26,66,'Ok'),
  button(145,10,20,60,'Cancel'),
  radio(70,10,20,130, 'colour : green', off, Green),
  radio(90,10,20,130, 'colour : red',off,Red),
  radio(110,10,20,130,'colour : black',on,Black),
  check(70,200,20,120,'left wheels',off,Left),
  check(90,200,20,120,'right wheels',off,Right),
  check(110,200,20,120,'sun roof',on,Roof),
  text(10,10,20,110,'Operator name:'),
  edit(10,125,20,95,'',Op),
  text(36,10,20,150,'Operator supervisor:'),
  edit(36,165,20,55,'Jones',read(Super))],
  Response).
```

- Also shows the *edit* field

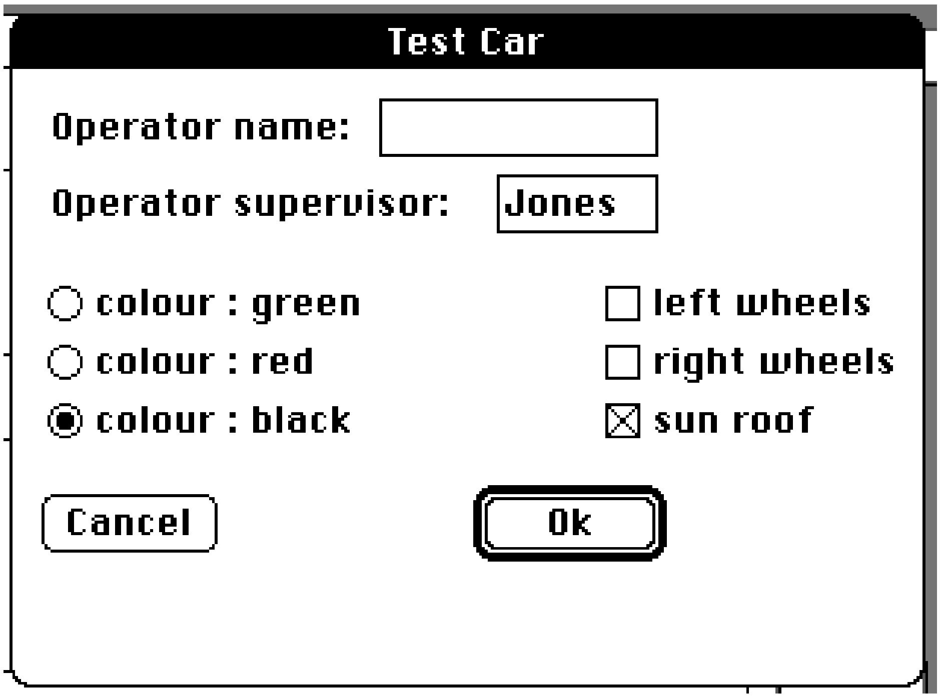

Figure 11: A Car panel

```
edit(Top,Left,Depth,Width,
  Initial Value, Final Value)
```

- *Edit field* is one where the user can enter a value - and is displayed as a rectangle
- *Initial Value* corresponds to the initial contents of the edit field, and *Final Value* corresponds to the variable holding the results of the user input
- Extension is

  ```
  scrolledit(Top, Left, Depth, Width,
    Initial, Final)
  ```

  which is a scrollable edit field in a dialog

```
altread :-
 prolog_flag(dialog_font,Font),
```

```
dialog('',50,60,120,370,
 [button(87,287,26,66,'OK'),
  button(90,10,20,60,'Cancel'),
  text(10,10,32,350,'Enter a term:',Font,10,0),
  edit(45,10,32,350,'',read(Term) ) ],
  Response).
```

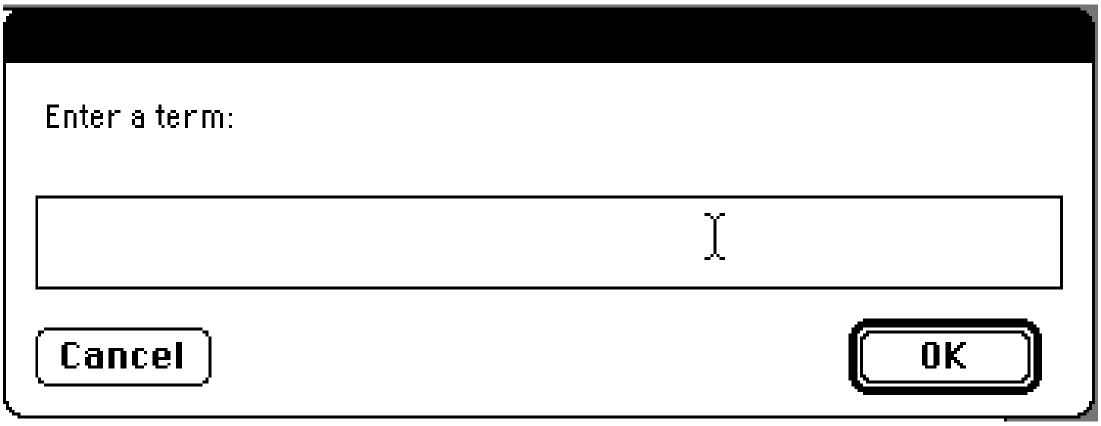

Figure 12: An alternate read dialog

Menus

```
menu(Top,Left,Depth,Width,Items,
    Pre selected, Selections)

menu(Top,Left,Depth,Width,Items,
    Pre selected, Selections,
    Font, Size, Style, Colour)

popup(Top,Left,Depth,Width,Label,
   Items, Pre selected, Selections)
```

- This described a scrolling menu in a dialog - where *Items* are the items in a menu

- *Pre selected* is one option highlighted when the menu is presented, and *Selections* are the choices made by the user
- It is also possible to include additional style commands like *Font, Size* and *Colour*
- The *popup* menu can be invoked from within a dialog also

```
calendar(SelDay,SelMonth) :-
 months(Months), days(Days),
 mdialog(100,220,150,200,
  [button(120, 130, 20, 60, 'Ok'),
   button(120,10,20,60,'Cancel'),
   text(8,8,16,180, 'Choose a day and month:'),
   popup( 35,10,20,165, 'Month:', Months,
   'March',SelMonth),
   popup(70,10,20,165, 'Day:  ',Days,
'Friday',SelDay)],
   Response).

months(['January', 'February', 'March',
   'April','May','June','July','August',
   'September','October','November','December']).

days(['Monday','Tuesday','Wednesday',
  'Thursday','Friday','Saturday','Sunday']).
```

5 Dialogs with Call backs built in

```
dialog(Title,Top,Left,Depth,Width,Items,
   Buttons, Goal)
```

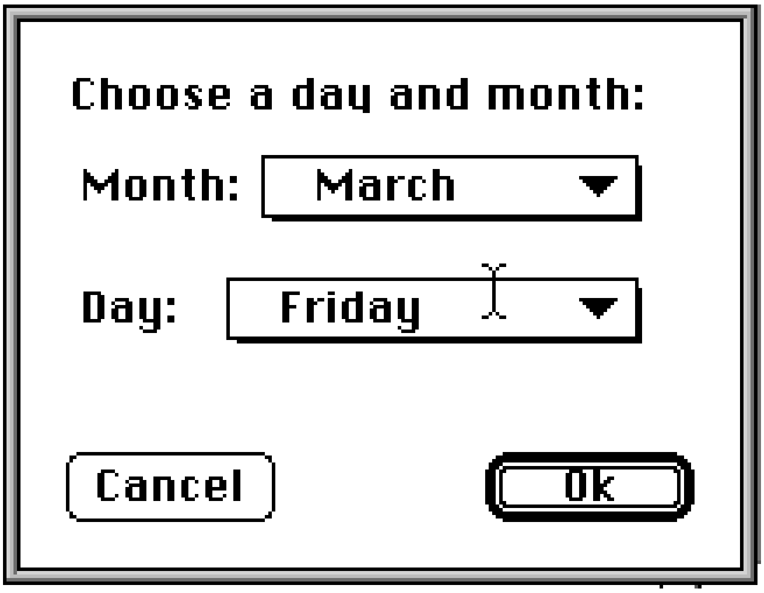

Figure 13: Calendar 1

- *Goal* specifies a program to be run when the *Ok* button (or any button other than Cancel) on the dialog is pressed
- If *goal* succeeds - dialog terminates
- If *goal* fails - dialog remains displayed and control returns to the user
- *Button* is an integer giving the number of the button that was pressed
- Consider the example - where the integers are expected - and if not entered - a warning message is displayed

```
readints(I1,I2) :-
 dialog('Integers',100,60,120,170,
 [button(65,97,26,66,'Ok'),
  button(68,10,20,60,'Cancel'),
  text(10,10,20,150,'Enter two integers:'),
  edit(34,12,20,70,'',read(I1)),
```

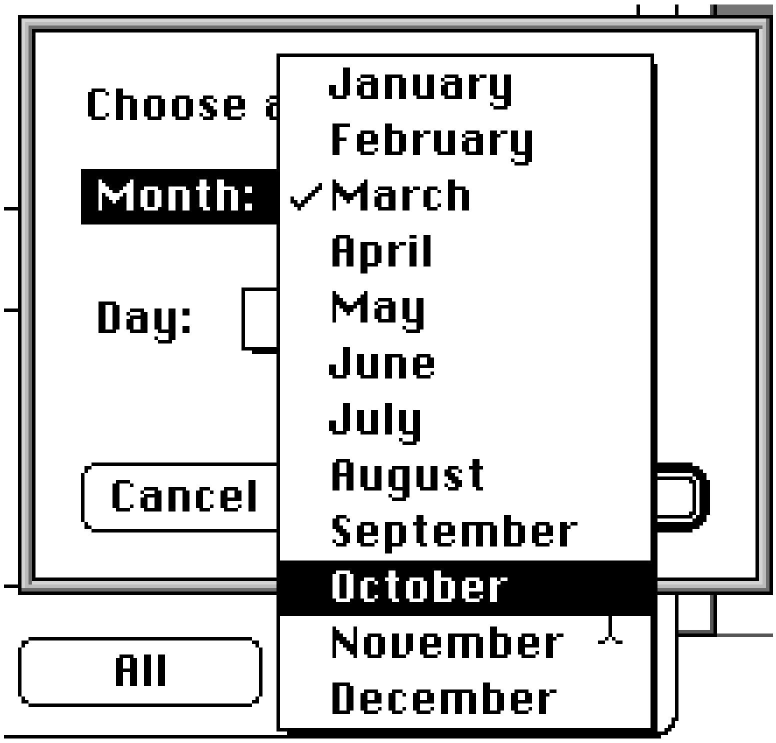

Figure 14:

```
  edit(34,90,20,70,'',read(I2))],
  Response,
  are_ints(I1,I2)).

are_ints(D, B, Int1, Int2) :-
 integer(Int1),
 integer(Int2), !.

are_ints(D, B, Int1, Int2) :-
 message(['You must enter integers']),
 fail.
```

5.1 Modal dialog

```
mdialog(Top,Left,Depth,Width,DItems,
```

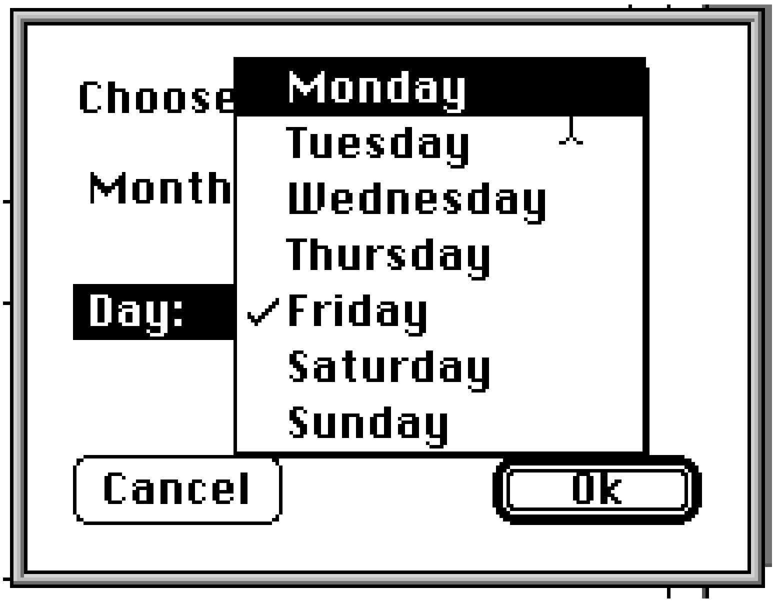

Figure 15:

```
  Button)

mdialog(Top,Left,Depth,Width,DItems,
           Button,Goal)
```

- *Modal* dialog - must be responded to before anything else is done

```
person(Ints,Occupation) :-
 mdialog(40,30,190,250,
 [button(157,177,26,66,'OK'),
  button(160,10,20,60,'Cancel'),
  text(15,20,20,40,'Age:'),
  edit(15,60,20,40,'21',Age),
  radio(7,150,20,80,'Male',off,Male),
  radio(23,150,20,80,'Female',on,Female),
  text(50,10,20,100,'Interests:'),
  text(50,140,20,100,'Occupation:'),
  menu(70,10,82,100,['Art','Badminton','Cinema',
```

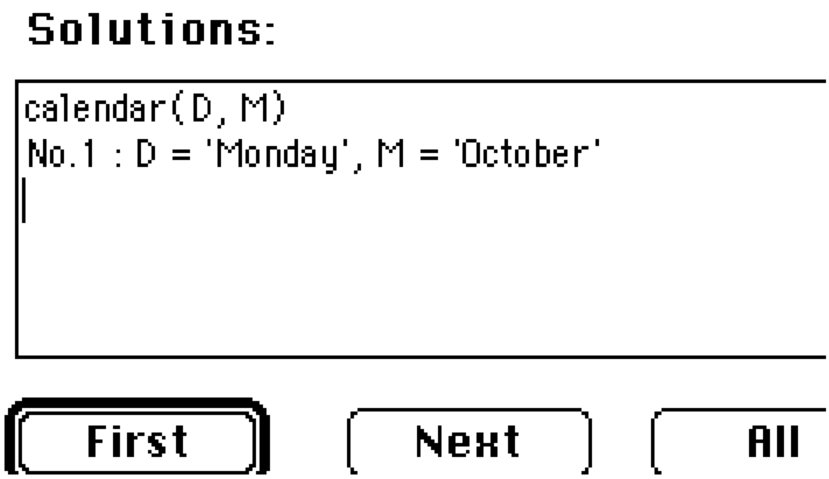

Figure 16:

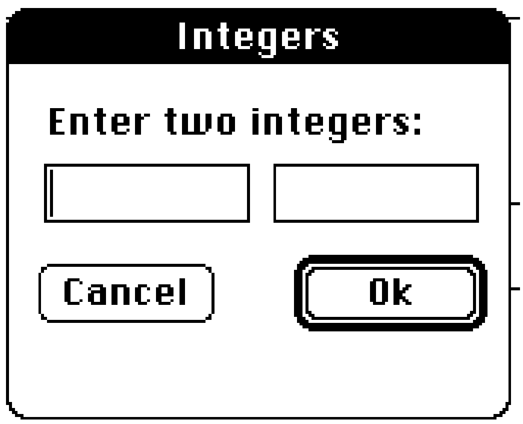

Figure 17: The integer dialog

```
  'Dancing','Eating','Fishing','Hiking'],
[],Ints, 'Courier',12,1),
menu(70,140,82,100,['Electrician','Gardner',
  'Scientist','Baker','Dressmaker','Baker',
  'Cab Driver'], 'Scientist',
Occupation, 'Courier',12,2,red)], Response).
```

- *Interests* menu is a multiple selections menu - with no items selected - hence the use of the *empty list*
- *Occupation* menu is a single selection menu - as no list is used in its specification

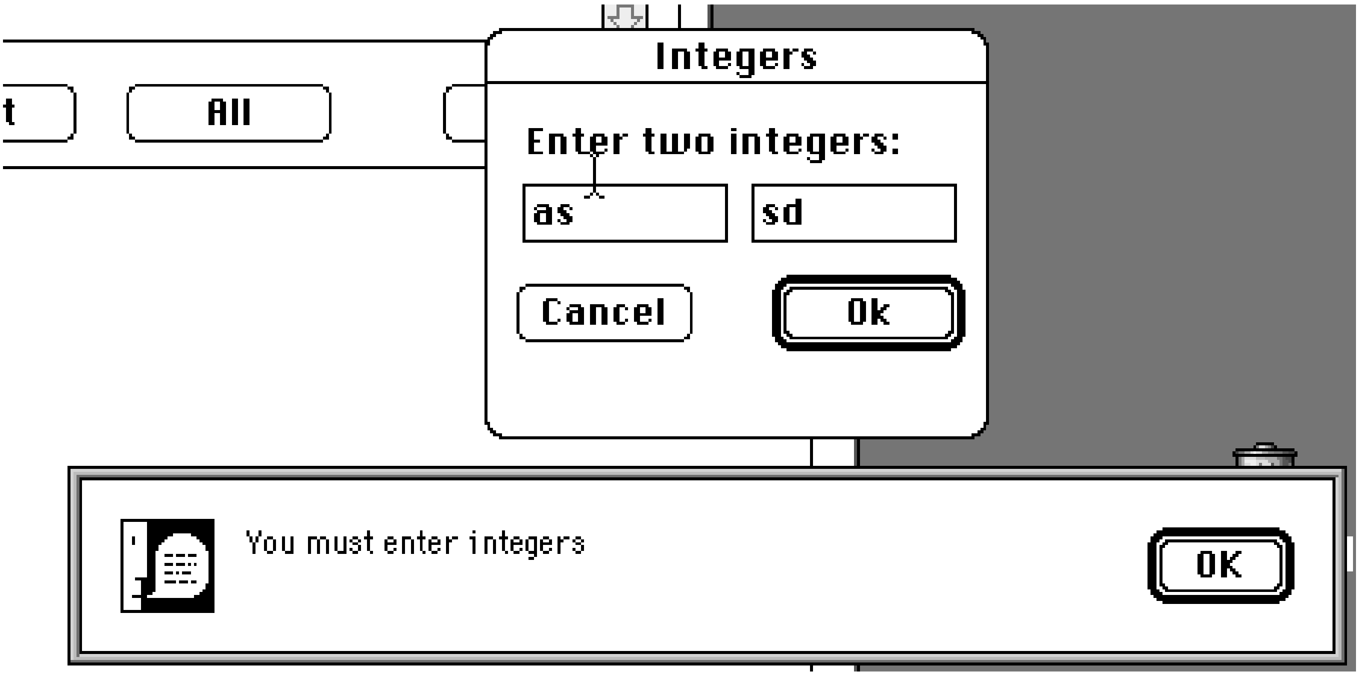

Figure 18: The integer dialog with error message

- Note also that the *occupation* menu appears in *red*

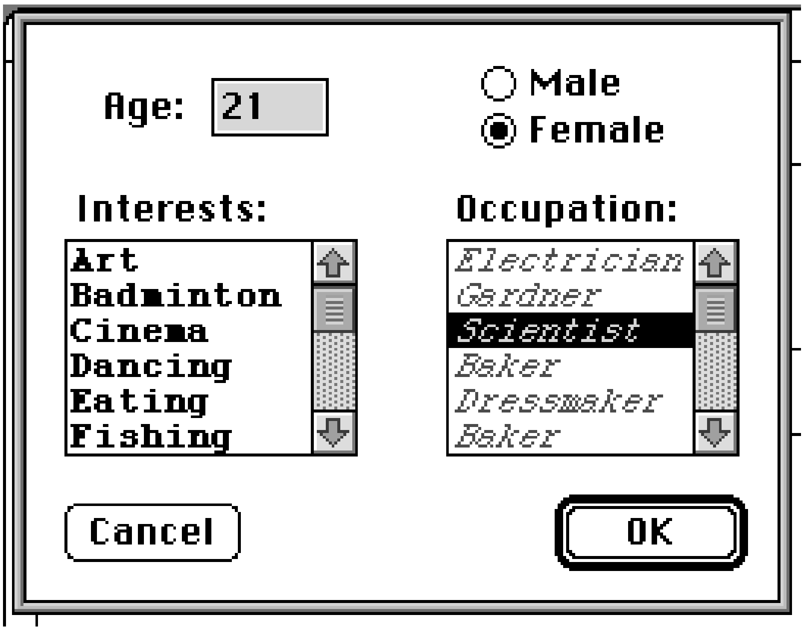

Figure 19: A modal dialog with menus