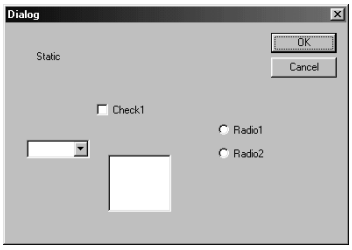
Dialog Boxes

- A dialog box is a type of window that provides a flexible means by which a user may interact with your program
- It interacts with the user through one or more **controls**.
- A control is a specific type of input or output window, which is owned by its parent window, the dialog box.

Controls

- ***Push buttons*** allow the user to activate a response by clicking on a button
- ***Check boxes*** allows items to be selected or unselected
- ***Radio buttons*** are essentially check boxes, but if there is more than one, only one may be selected, thus allowing mutual exclusion.
- A ***list box*** displays a list of items from which the user selects (e.g. file names).
- ***Edit boxes*** allow the user to enter a string.
- ***Combo boxes*** combine a list box with an edit box
- ***Scroll bars*** allow text in a window to be scrolled
- ***A static control*** allows text to be output, but accepts no input.



Controls

- Controls generate messages when accessed by a user.
- They also receive messages from your application.
- A message generated by a control indicates what type of interaction the user has had with the control.
- A message sent to the control is an instruction to which the control must respond.

The Control Classes

- MFC defines classes for the common controls, derived from CWnd.
- Thus, a control has access to the same general functionality that is available to any other type of window.
- The classes for the standard controls are: CButton, CEdit, CListBox, CComboBox, CScrollBar, Cstatic.

The Control Classes

- Although MFC allows you to access the control directly, it is advisable to access them through a control object.
- For simple uses of the push button, this may not be required, but for more sophisticated manipulation, it should be encapsulated.
- It is also possible to create free-standing controls within your program's main window.
  - They are, after all, just special types of windows.

Modal vs. Modeless Dialog Boxes

- The most common dialog boxes are modal
  - I.e. they demand a response before the program will continue
- A modeless dialog box does not prevent the parent program from running.

The Dialog Box

- A dialog box is simply another resource that is contained in your program's resource file.
- You create this, as before, using a resource editor.
- Dialog boxes are encapsulated by the CDialog class, which is derived from CWnd.
- In MFC, all dialog boxes are objects that are instances of either CDialog itself, or a class derived from it.

The Dialog Box Class

- Only the most trivial dialog boxes are objects of CDialog, and these are better implemented as message boxes.
- In general, when your program requires a dialog box, you will derive your own dialog box from Cdialog.
- In this way, you can add the extra functionality that describes how your program reacts to a variety of sophisticated controls.

Processing Dialog Box Messages

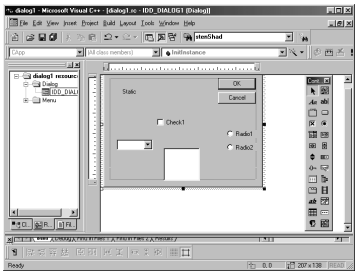
- Since your dialog box is ultimately derived from CWnd, all dialog boxes are windows.
- Event that occur within a dialog box are sent to your program using the same message-passing mechanism the main window uses.
- However, dialog box messages are not sent to your program's main window, and do not use its message map.
- Instead, each message map that you define will need its own message map and handlers.

Processing Dialog Box Messages

- Each control within a dialog box will be given its own resource ID.
- Each time that control is accessed by the user, a WM_COMMAND message will be sent to your dialog box window.
- Associated with that message will be the id of the control that was accessed.
- The ON_COMMAND message macro can be used to decode and handle each message, in a similar way to menus.
  - However, unlike menu messages, the dialog box messages are sent to the dialog box's message map, not to the main window's message map.

Using a Dialog Box -1

- Create the dialog box resource, adding any controls you need:



Using a Dialog Box -2

- Derive a class from CDialog (in the include file) and declare a separate message map for the dialog box.

```
class CSampleDialog : public CDialog
{
public:
   CSampleDialog(char *DialogName, CWnd *Owner)
                  : CDialog(DialogName, Owner){}

   DECLARE_MESSAGE_MAP()
};
```

Using a Dialog Box -3

- Instantiate an object of your derived dialog class (in the source file), then activate it.

```
CSampleDialog diagOb(
          MAKEINTRESOURCE(IDD_DIALOG1),this);
diagOb.DoModal();
          //activate modal dialog box
```

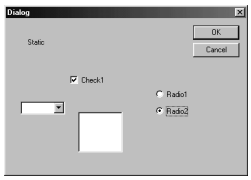
Using a Dialog Box -4

- Create a message map for the dialog box (in the source file).

```
//This is the CSampleDialog's message map
BEGIN_MESSAGE_MAP(CSampleDialog, CDialog)
END_MESSAGE_MAP()
```

Using a Dialog Box

- This will display the dialog box, but nothing interesting will happen in response to manipulation of the controls.



Reacting to Push Buttons

- The next step is to react to user manipulation of the controls.
- For each control, we need to add a handler to the dialog box class, and to the message map.
- E.g. in the include file:

```
class CSampleDialog : public CDialog
{
public:
   CSampleDialog(char *DialogName, CWnd *Owner) :
                  CDialog(DialogName, Owner){}
   afx_msg void OnMyButton();
   DECLARE_MESSAGE_MAP()
};
```

Reacting to Push Buttons

- In the Source file:

```
afx_msg void CSampleDialog::OnMyButton(){
      MessageBox("My Button","Button Selected");
}
```

```
//This is the CSampleDialog's message map
BEGIN_MESSAGE_MAP(CSampleDialog, CDialog)
      ON_COMMAND(IDC_BUTTON1, OnMyButton)
END_MESSAGE_MAP()
```

OnOK() and OnCancel()

- The buttons ok and cancel are added automatically by the resource editor.
- These two buttons are so common, that there is a default handler for each:
  - CDialog::OnOK() and CDialog::OnCancel()
- These built-in functions are not added to the dialog box message map
- Their default behaviour is to close the dialog box.
- You can over-ride them by providing your own version of the handler

E.g. Over-riding OnCancel()

```
afx_msg void CSampleDialog::OnCancel()
{
int response;
response = MessageBox("Are you sure?",
                      "Cancel",MB_YESNO);
if(response==IDYES)EndDialog(0);
}
```

- In order to use this over-ridden handler, you must:
  - add its prototype to your dialog box class definition
  - Add the following message handler to the dialog box message map:

```
ON_COMMAND(IDCANCEL, OnCancel)
```

Initializing a Dialog Box

- Sometimes you will need to initialize various variables or controls associated with a dialog box before it is displayed.
- To allow a dialog box to perform these initializations, Windows automatically send it a WM_INITDIALOG message when the dialog box is first dreated.
- When this message is received, MFC automatically calls OnInitDialog(), which is a built-in message handler defined by CDialog.

Initializing a Dialog Box

- You may override OnInitDialog() to provide specialized functionality upon initialization of the dialog box.
- There is one important constraint…
  - Your implementation of OnInitDialog() must call CDialog::OnInitDialog() as the first thing it does.

Adding a List Box

- List boxes are managed in an MFC program through the use of the CListBox class.
- A list box will both generate and receive messages.
- For example, you will send a list box messages when initializing it, by sending it the list of strings that it will display.
- By default, the list box is empty.
- Once the list box has been initialized, it will generate messages in response to user actions.

Receiving List Box Notifications

- A list box generates various types of messages. E.g
  - It will generate a message when an item in the box has been double-clicked,
  - When the box is losing input focus
  - When the selection inside the box is being changed.
- These events are describe by their corresponding notification code.
- A notification code is part of a WM_COMMAND message, and is essentially a secondary message that describes the event.

Example: double-click notification

- The notification code LBN_DBLCLK (list box notification – double click) is sent when the user has double-clicked on an entry in the list.
- Once a selection has been made, you must then query the list box to find out which item has been selected.
- To process the LBN_DBLCLK notification message, you must add its message handler to your message map.

Example: double-click notification

- However, you do not use ON_COMMAND to process it.
- Instead, you use a special notification message macro to handle notification codes.
- All list box notification macros begin with ON_LBN, so the double click macro is:
  - ON_LBN_DBLCLK(ID, HandlerName)
  - where ID is the resource id of the list box, and HandlerName is the name of the function that processes that ID.

Sending List Box Messages

- There are several different messages that can be sent to a list box.
- Thus, the CListBox class contains several member functions that send these messages.
- E.g.
  - int CListBox::AddString(*string*);
  - int CListBox::GetCurSel();
  - int CListBox::GetText(*index, string*);

- AddString(*string*)
  - inserts the specified string inito the list box.
  - By default, the string is inserted at the end of the current list.
  - It returns the index of the string in the list. (List box indices start at 0)
- GetCurSel()
  - returns the index of the currently highlighted/selected item in the list.
- GetText(*index, string*)
  - Copies the text associated with the specified index into string.
  - It returns the number of characters in the string.

Pointer to a List Box

- To use a list box defined as a resource within a resource file, you must first obtain a pointer to it, using GetDlgItem().
- GetDlgItem(*id*) returns a pointer to the object with resource identifier id.
- This is a temporary pointer, so you must obtain it every time your program needs it,
  - You can't store it in a global variable or class data member, for instance.

Pointer to a List Box

- When obtaining a pointer to a control object, the pointer returned must be cast into a pointer to the type of object being obtained.
  - I.e. a pointer to a list box object must be cast into a CListBox * pointer.
- Since a list box is empty by default, you must initialize it each time the dialog box that contains it is displayed.
- To do this, you implement your own version of OnInitDialog(), which adds strings to the box.

Initializing the list box

```
BOOL CSampleDialog::OnInitDialog()
{
Cdialog::OnInitDialog();//call base class version
CListBox *lbPtr = (CListBox *)GetDlgItem(IDD_LB1);

//Add strings to list box
lbPtr->AddString("Red");
lbPtr->AddString("Green");
lbPtr->AddString("Blue");
lbPtr->AddString("Pink");

return TRUE;
}
```