

8. Casete de dialog modale și nemodale

În marea majoritate a aplicațiilor, o acțiune asupra butoanelor implică apariția unor noi casete de dialog, numite casete copil, care la rândul lor vor conține o serie de obiecte de interfață. Aceste casete, pot lansa apoi alte casete și așa mai departe.

Pentru fiecare casetă suplimentară inserată în proiect, trebuie adăugată o resursă de tip casetă de dialog și o clasă asociată, derivată din clasa `CDialog`. Casetă de dialog poate fi creată prin intermediul editorului de resurse (**ClassView**), iar clasa derivată este creată de către **ClassWizard**. Noua clasă poate fi dezvoltată pentru a asigura gestiunea controalelor aflate în spațiul noii machete.

Marea majoritate a casetelor de dialog copil sunt lansate în execuție ca și casete de dialog modale. La afișarea unei astfel de casete, restul interfeței cu utilizatorul devine inaccesibil, fiind necesară închiderea casetei înainte de a relua lucrul obișnuit. Casetele de dialog pot fi cascade, astfel încât o casetă să apeleze pe alta, determinând în acest fel transmiterea controlului către caseta de dialog cel mai recent deschisă, care este de altfel și caseta activă. După închiderea acesteia, se redă controlul casetei de dialog apelante.

Există și posibilitatea lansării casetelor de dialog ca și casete de dialog nemodale, ca alternativă la casetele de dialog modale. Aceste casete permit accesul și la restul interfeței în timpul în care sunt afișate. În fapt, o serie de controale sunt implementate ca și casete de dialog nemodale.

8.1 Cum adăugăm o nouă casetă de dialog

Pentru a vedea cum se poate adăuga nouă casetă de dialog proiectului, să creem un nou proiect **MFC AppWizard (.exe)** de tip `Dialog Based`, numit *wiz5*. Casetă de dialog asociată acestui proiect o vom configura ca în fig. 8.1.

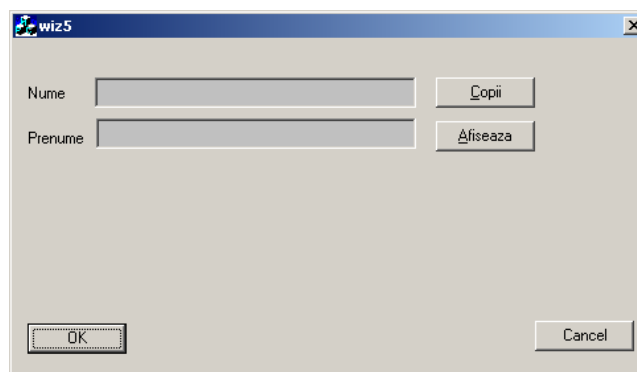


Figura 8.1. Casetă de dialog părinte

Am adăugat casetei de dialog casetele de editare (de jos în sus) `IDC_NUME` și `IDC_PRENUME` și butoanele cu identificatorii `IDC_COPII` și `IDC_AFISEAZA`, având respectiv etichetele Copii și Afiseaza. De asemenea, am mapat casetelor de editare, respectiv variabilele de categorie `value`, `CString m_strNume` și `CString m_strPrenume`.

Să implementăm funcția ce răspunde apăsării butonului `IDC_AFISEAZA`:

```

void CWiz5Dlg::OnAfiseaz()
{
    // TODO: Add your control notification handler code here
    UpdateData();
    CString strText;
    strText += m_strNume+"    " + m_strPrenume;
    MessageBox(strText,"Nume parinte",MB_ICONEXCLAMATION);
}

```

Va trebui acum să adăugăm o nouă casetă de dialog proiectului nostru. Pentru aceasta, vom efectua următoarele:

- vom selecta **ResourceView** pentru a afișa resursele proiectului;
- apăsăm click stânga pe **wiz5 resources** și apoi se apăsăm click dreapta pentru a afișa meniul contextual;
- alegem opțiunea **Insert...**, pentru a deschide caseta de dialog **Insert Resource** (fig. 8.2).

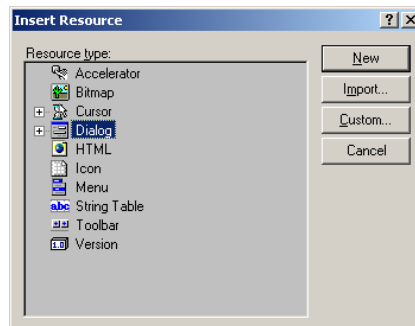


Figura 8.2. Inserăm o nouă casetă de dialog

- selectăm **Dialog** pentru a arăta că dorim inserarea unei casete de dialog ;
- apăsăm butonul **New** și noua resursă de tip casetă de dialog este adăugată proiectului. Ea trebuie să apară acum în **ResourceView** ca subordonată (de tip **Dialog**) lui **wiz13 resources**. Ea are implicit identificatorul **IDD_DIALOG1**. Dorim să schimbăm acest identificator;
- apăsăm click dreapta pentru afișarea meniului contextual și alegem **Properties**;
- modificăm identificatorul noii casete în **IDD_CASETAMODALA**;
Urmează să adăugăm controale casetei de dialog. Vom adăuga controale ca și în fig. 8.3:

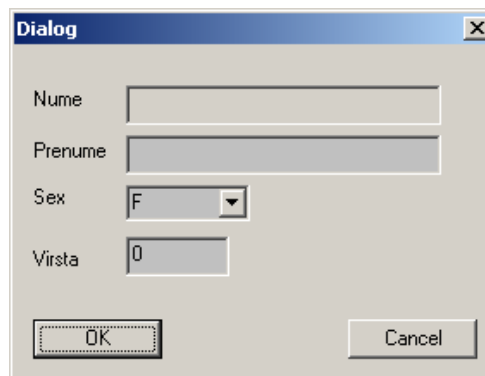


Figura 8.3 Casetă de dialog modală

Am adăugat casetei, 3 casete de editare și o casetă combinată. Prima casetă de editare are identificatorul **IDC_NUME** și stilul **Read_only** validat. A doua casetă de

editare are identificatorul `IDC_PRENUME`, iar a treia `IDC_VIRSTA`. Caseta combinată are identificatorul `IDC_SEX` și stilul `Sort` nevalidat.

Dorim acum să mapăm controale variabilelor. Dacă lansăm **ClassWizard**, aceasta va sesiza existența unei noi resurse de tip dialog și va lansa un dialog prin intermediul căruia va asocia o clasă (implicit derivată public din `CDialog`) noii resurse. Pentru aceasta:

- apăsăm click pe noua resursă pentru a o selecta (sau dublu click pe `IDD_CASETAMODALA` din **ResourceView**);
- lansăm **ClassWizard**;
- **ClassWizard** va sesiza că am inserat o nouă casetă de dialog va afișa caseta **Adding a Class** (fig. 8.4);

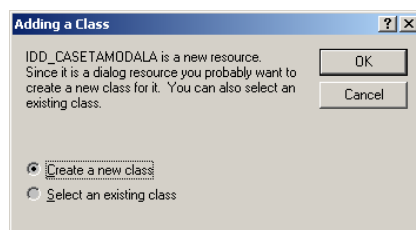


Figura 8.4. Adăugăm o nouă clasă

- apăsăm **OK** pentru acceptarea opțiunii implicite de creare a unei noi clase;
- **ClassWizard** va afișa caseta de dialog **New Class** (fig. 8.5);

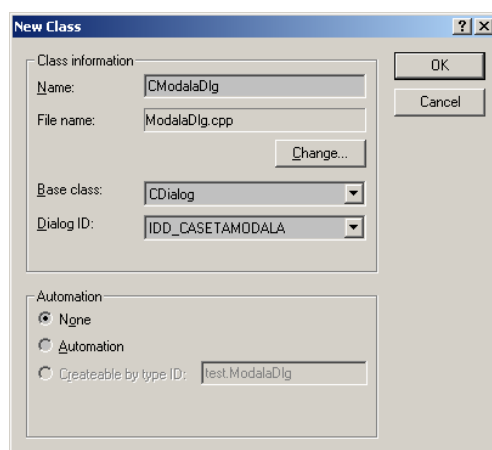


Figura 8.5. Așa definim noua clasă

- în caseta de editare **Name**: vom specifica numele noii clase, care încapsulează noua casetă de dialog. Prin convenție, clasele derivate din clasele MFC încep cu litera **C**. Fie **CModalDlg** numele acestei noi clase;
- o dată cu tastarea numelui pentru clasa derivată, automat, în caseta de editare **File Name**: va apare numele fișierului care conține definiția clasei. Acest nume este implicit similar cu numele clasei, deci, în cazul nostru, *ModalDlg.cpp*. Acest nume poate fi schimbat de către utilizator;
- în caseta combinată **Base class**: este afișată superclasa din care este derivată noua clasă. Aceasta, pentru o clasă care implementează o casetă de dialog, este implicit `CDialog`;
- în caseta combinată **Dialog ID**: este afișat identificatorul noii casete de dialog. Și acesta poate fi schimbat, în acest caz clasa referindu-se la altă casetă;
- apăsăm **OK** pentru crearea noii clase, scheletul acesteia fiind creat automat în fișierele *.h* și *.cpp*;

- **ClassWizard** va afișa apoi pagina **Message Maps** obișnuită, putând fi adăugate funcții de tratare a evenimentelor din noua casetă de dialog; Noi vom trece în pagina **Member Variables** și vom asocia controalelor următoarele variabile (după ce ne convingem că în caseta combinată **Class name:** este trecut **CModalaDlg**): casetelor de editare **IDC_PRENUME** și **IDC_VIRSTA** variabilele de categorie **Value** **CString m_strPrenume** și **int m_nVirsta** și casetei combinate **IDC_SEX** variabila de categorie **Control**, **CComboBox m_cbSex**.
- apăsăm **OK** pentru închiderea **Class Wizard**;

După adăugarea noii clase care încapsulează caseta de dialog, aceasta va apare în **Class View**. Toate funcțiile asociate evenimentelor generate de controalele de pe caseta de dialog copil și respectiv toate variabilele mapate acestora, vor aparține acestei clase. Implicit, pentru această clasă sunt create constructorul și o funcție de schimb de date. Constructorul clasei are implementarea:

```
CModalaDlg::CModalaDlg(CWnd* pParent /*=NULL*/)
: CDialog(CModalaDlg::IDD, pParent)
{
   //{{AFX_DATA_INIT(CModalaDlg)
    //}}AFX_DATA_INIT
}
```

Constructorul primește ca parametru un pointer spre clasa **CWnd** și construiește noua casetă apelând constructorul clasei de bază, **CDialog**, căruia îi transmite un identificator, definit în clasa derivată, **CModalaDlg::IDD**. Acest identificator este declarat în definiția clasei (fișierul *ModalaDlg.h*) ca o valoare de tip enumerare și inițializat cu identificatorul de resursă al casetei de dialog:

```
...
class CModalaDlg : public CDialog
{
// Construction
public:
    CModalaDlg(CWnd* pParent = NULL);    // standard constructor
// Dialog Data
    {{{AFX_DATA(CModalaDlg)
    enum { IDD = IDD_CASETAMODALA }; ...
    ...
}
```

Observație: În cazul în care proiectul conține mai multe clase de dialog, este necesară acordarea unei atenții sporite în cazul inserării de noi variabile cu ajutorul **ClassWizard**. Trebuie ca să fie selectată clasa potrivită în caseta **Class name:**, în caz contrar variabila fiind asociată unei alte clase.

Noua casetă de dialog este apelată ca răspuns la o acțiune asupra unui buton din caseta părinte, sau la selectarea unui meniu. Deci, în funcția de răspuns la evenimentul din caseta părinte, va trebui realizată și afișarea casetei de dialog derivate. Vom realiza aceasta în doi pași:

- vom crea o instanță a clasei dialog, declarând un obiect local funcției de tratare, ca de exemplu

```
CModalaDlg dlgCasetamodala(this);
```

Ce facem de fapt? Am declarat un obiect de clasa `CModalaDlg`. Să ne uităm câteva rânduri mai sus și să vedem că trebuie să-i transmitem constructorului un pointer `CWnd`, mai clar spus, un pointer spre caseta de dialog părinte. Cum în acest moment aceasta este caseta activă, vom transmite constructorului pointerul `this` (vă reamintiți?).

Pentru ca procesul de compilare să recunoască clasa de dialog derivată, trebuie ca fișierul header de definiție a clasei să fie inclus la începutul modulului care conține instanțierea. De exemplu, în cazul nostru, dacă definiția noii clase se află în *ModalaDlg.h*, în fișierul *wiz5Dlg.cpp* care instanțiază această clasă, la început va fi inserată linia

```
/ wiz5Dlg.cpp : implementation file
//

#include "stdafx.h"
#include "wiz5.h"
#include "wiz5Dlg.h"
#include "ModalaDlg.h"

#ifdef _DEBUG
...
```

- vom afișa caseta și vom declanșa operarea modală a casetei de dialog. Acest lucru se face prin apelul funcției `DoModal()`. Ca efect, va fi afișată caseta de dialog împreună cu controalele sale componente și va fi iterată o buclă de mesaje locale, mesajele fiind transmise de la controalele casetei de dialog către implementarea utilizator, până la închiderea casetei de dialog.

Dacă apare o eroare la afișarea noii casete de dialog, `DoModal()` returnează un cod de eroare, care poate fi `-1` sau `IDABORT`. Dacă afișarea casetei reușește, din `DoModal()` se revine la apelul implicit al funcției `EndDialog()`. Funcția `EndDialog()` primește ca parametru o valoare întreagă, care specifică cine a apelat-o, valoare întoarsă apoi funcției `DoModal()` care o utilizează ca și cod de retur.

Funcția `EndDialog()` este apelată în implementările implicite ale funcțiilor `OnOK()` și `OnCancel()`, lansate în execuție la producerea evenimentului `BN_CLICKED` pentru butoanele **OK** și **Cancel**. În acest caz, ea va transmite respectiv codurile `IDOK` și respectiv `IDCANCEL` funcției `DoModal()`.

Caseta de dialog poate fi închisă de utilizator în orice moment și prin program, prin apelarea funcției `EndDialog()`.

Implementarea funcției `OnCopii()` este:

```
void CWiz5Dlg::OnCopii()
{
    // TODO: Add your control notification handler code here
    CModalaDlg dlgCasetaModala(this);
    int nRetCode;
    UpdateData();
    if (!m_strNume.IsEmpty())
    {
        dlgCasetaModala.m_strNume=m_strNume;
    }
}
```

```

        nRetCode = dlgCasetaModala.DoModal();
    }
    else
        MessageBox("Nu avem un nume de parinte!",
            "Eroare", MB_ICONSTOP);
}

```

Am definit caseta derivată care va fi identificată prin `dlgCasetaModala`, fiind apoi afișată. Se pot face unele inițializări a anumitor variabile, presupunând de exemplu că numele copilului este același cu al părintelui și că acesta a fost completat. Inițializarea se face înainte de afișarea casetei derivate.

Inițializările mai complexe, cum ar fi popularea casetei combinate, se fac altfel. Se poate observa că la compilare, o linie de forma

```
dlgCasetaDerivata.m_cbSex.AddString("F");
```

nu va da eroare de compilare, dar va da eroare de aserție la execuție. Să ne reamintim că astfel de inițializări se făceau, pentru fereastra părinte, în funcția `OnInitDialog()`. Dacă ne uităm în implementarea casetei derivate, constatăm că această funcție nu există. Dar, ea poate fi implementată, ca funcția ce răspunde mesajului `WM_INITDIALOG` pentru caseta de dialog modală. Deci, o vom putea crea cu ajutorul **Class Wizard**, având grijă ca la **Class name**: să fie selectat `CModalaDlg`. Și la **Object IDs**: `CWiz5Dlg`. Implementarea funcției de inițializare va fi:

```

BOOL CModalaDlg::OnInitDialog()
{
    CDialog::OnInitDialog();

    // TODO: Add extra initialization here
    PopulezCombo();
    return TRUE; // return TRUE unless ...
}

```

Vom adăuga clasei `CModalaDlg` funcția membri `void PopulezCombo()` pentru inițializarea valorilor casetei combinate `IDC_SEX`: Vom adăuga tot clasei `CModalaDlg` variabila membru `int nIndex`.

```

void CModalaDlg::PopulezCombo()
{
    m_cbSex.AddString("F");
    m_cbSex.AddString("M");
    int nIndex=m_cbSex.SelectString(-1,"F");
}

```

Va trebui acum să preluăm textul selectat în caseta combinată. Pentru aceasta, vom adăuga clasei `CModalaDlg` variabila membru `CString m_strSex` și vom implementa funcția ce răspunde mesajului `CBN_SELCHANGE`:

```

void CModalaDlg::OnSelchangeSex()
{
    // TODO: Add your control notification handler code here
    int nIndex=m_cbSex.GetCurSel();
    if (nIndex != CB_ERR)
        m_cbSex.GetLBText(nIndex, m_strSex);
}

```

Va trebui să modificăm funcția `PopulezCombo()`, astfel încât șirul `m_strSex` să fie actualizat și pentru valoarea implicită din caseta combinată:

```
void CModalaDlg::OnSelchangeSex()
{
    ...
    if (nIndex != CB_ERR)
        m_cbSex.GetLBText(nIndex, m_strSex);
}
```

8.2 Schimbul și validarea datelor

Atunci când se mapează variabile controalelor, **ClassWizard** generează automat cod pentru efectuarea schimbului de date între controale și variabilele asociate, prin intermediul funcției `DoDataExchange()` a clasei de dialog. Orice clasă de dialog va conține o astfel de funcție. Ea răspunde de transferul datelor în ambele direcții, înspre și dinspre controalele casetei de dialog.

Procesul de transfer se numește *schimb de date* și este inițiat printr-un apel al funcției `UpdateData()`. Această funcție are ca parametru variabila `BOOL bSaveAndValidate`, care ia implicit valoarea `TRUE`. Să ne reamintim că valoarea acestei variabile dictează sensul transferului informației: dacă este `FALSE`, controalele sunt actualizate pe baza conținutului variabilelor asociate, iar dacă este `TRUE`, variabilele iau valoarea înscrisă în controale. Funcția `UpdateData()` este apelată automat de funcția `OnInitDialog()`, cu parametrul `FALSE` pentru a inițializa controalele la afișarea casetei de dialog. Funcția `OnOK()` apelează implicit `UpdateData()` cu parametrul `TRUE` pentru actualizarea variabilelor asociate cu conținutul controalelor.

Există două tipuri de funcții pentru efectuarea transferului de date.

8.2.1 Funcții pentru schimbul de date DDX (Do Data Exchange)

Funcția `DoDataExchange()` pentru caseta modală din exemplul `wiz5` este:

```
void CModalaDlg::DoDataExchange(CDataExchange* pDX)
{
    CDialog::DoDataExchange(pDX);
   //{{AFX_DATA_MAP(CModalaDlg)
    DDX_Control(pDX, IDC_SEX, m_cbSex);
    DDX_Text(pDX, IDC_NUME, m_strNume);
    DDX_Text(pDX, IDC_PRENUME, m_strPrenume);
    DDX_Text(pDX, IDC_VIRSTA, m_nVirsta);
   //}}AFX_DATA_MAP
}
```

Funcția primește ca parametru un pointer la clasa `CDataExchange`. Această clasă implementează toate detaliile referitoare la transferul de date, inclusiv referitor la fereastra vizată de transfer. De altfel și variabila `bSaveAndValidate` este o variabilă membru a acestei clase. Pointerul `pDX` este apoi utilizat în apelul funcțiilor `DDX_Text` sau `DDX_Control` (generate de **ClassWizard**, în funcție de tipul variabilei care mapează controlul) în scopul implementării transferului de date. Există mai multe

astfel de funcții `DDX_`, care răspund la diferite tipuri de controale și de date. Funcțiile `DDX_` specifică modul efectiv de executare a transferului. De exemplu, linia `DDX_Text(pDX, IDC_NUME, m_strNume);` se citește astfel: se execută transferul de date de tip șir de caractere, în sensul indicat de `pDX`, între controlul `IDC_NUME` și variabila `m_strNume`. Există mai multe categorii de funcții `DDX_` de transfer de date de tip valoare, prezentate în tabelul 8.1

Tabelul 8.1

Nume	Tipuri de control	Tipuri de date asociate
<code>DDX_Text</code>	Casetă de editare	<code>BYTE</code> , <code>short</code> , <code>int</code> , <code>UINT</code> , <code>long</code> , <code>DWORD</code> , <code>CString</code> , <code>LPTSTR</code> , <code>float</code> , <code>double</code> , <code>COleCurrency</code> , <code>COleDateTime</code>
<code>DDX_Check</code>	Casetă de validare	<code>int</code>
<code>DDX_Radio</code>	Grup de butoane de opțiuni	<code>int</code>
<code>DDX_LBString</code>	Casetă cu listă derulantă	<code>CString</code>
<code>DDX_LBStringExact</code>	Casetă cu listă derulantă	<code>CString</code>
<code>DDX_CBString</code>	Casetă combinată	<code>CString</code>
<code>DDX_CBStringExact</code>	Casetă combinată	<code>CString</code>
<code>DDX_LB_Index</code>	Casetă cu listă derulantă, val. de index	<code>int</code>
<code>DDX_CBIndex</code>	Casetă combinată, val. de index	<code>int</code>
<code>DDX_Scroll</code>	Bară de derulare	<code>int</code>

Implicit, aceste funcții sunt adăugate automat de **ClassWizard**, dar pot fi adăugate și de utilizator, manual, în funcția `DoDataExchange()`, respectând aceleași reguli cu **ClassWizard**.

Dacă dorim să transferăm valori între controale și variabilele asociate, va trebui să utilizăm funcția `UpdateData()`. Dar, am observat că, în exemplul cu caseta modală, numele s-a transmis acesteia fără să fie nevoie să apelăm `UpdateData()`. Este normal, să ne reamintim că `OnInitDialog()` apelează implicit această funcție, cu parametrul `FALSE`. De asemenea, să ne reamintim că `OnOK()` apelează aceeași funcție, cu parametrul `TRUE`. Deci, dacă vom utiliza valorile înscrise în controalele casetei de dialog copil doar în caseta de dialog părinte, transferul de date în variabile se face implicit, fără să mai apelăm `UpdateData()`.

8.2.2 Funcții pentru validare de date DDV (Do Data Validate)

Există anumite situații, în care trebuie realizate validări ale variabilelor ce preiau conținutul unei casete de dialog. De exemplu, unei variabile `CString` mapată peste o casetă de editare, i se poate preciza numărul maxim de caractere din șirul introdus de utilizator. În cazul unei variabile `int` de exemplu, se pot preciza limitele superioară și inferioară pentru domeniul în care variabila ia valori.

La adăugarea unei reguli de validare, **ClassWizard** va genera linii `DDV_` corespunzătoare în funcția `DoDataExchange()`. Dacă de exemplu, `IDC_VIRSTA` poate lua valori doar între 0 și 18 (precizate în câmpurile **Minimum Value:** și **Maximum Value:** din pagina **Member Variables** al **ClassWizard**), această funcție va fi

```
void CModalDlg::DoDataExchange(CDataExchange* pDX)
{
    CDialog::DoDataExchange(pDX);
}
```



```

...
DDX_Text(pDX, IDC_VIRSTA, m_nVirsta);
DDV_MinMaxInt(pDX, m_nVirsta, 0, 18);
//}}AFX_DATA_MAP
}

```

Se observă apariția liniei `DDV_MinMaxInt(pDX, m_nVirsta, 0, 18);` care specifică faptul că variabila `m_nVirsta` ia valori între 0 și 18. Există mai multe categorii de funcții `DDV_` standard, prezentate în tabelul 8.2:

Tabelul 8.2

Funcția DDV	Tip de date	Descriere
DDV_MaxChars	CString	Limitează nr. de caractere introduse
DDV_MinMaxByte	BYTE	Limitează nr. la un interval specificat
DDV_MinMaxDouble	double	Limitează nr. la un interval specificat
DDV_MinMaxDword	DWORD	Limitează nr. la un interval specificat
DDV_MinMaxFloat	float	Limitează nr. la un interval specificat
DDV_MinMaxInt	int	Limitează nr. la un interval specificat
DDV_MinMaxLong	long	Limitează nr. la un interval specificat
DDV_MinMaxUnsigned	unsigned	Limitează nr. la un interval specificat

Se pot crea funcții de validare proprii, care să efectueze verificări utilizator, construind funcții `DDV_` personalizate. În acest scop, trebuie creată o funcție care primește un pointer la clasa `CDataExchange`, o referință la variabila mapată care va fi testată și orice alți parametri specifici.

Prima condiție pe care trebuie să o verifice funcția este dacă obiectul `CDataExchange` se află în modul *salvează-și-validează*. Acest lucru se întâmplă dacă `pDX->m_bSaveAndValidate=TRUE`. Dacă acest indicator nu este `TRUE`, atunci se efectuează inițializarea controalelor pe baza variabilelor membru, așa că nu sunt necesare teste de validare și funcția ar trebui să-și încheie execuția.

Dacă variabila mapată este validă, funcția se încheie normal. În caz contrar, se efectuează o acțiune de eroare, care poate afișa o casetă de avertizare, sau se pot întreprinde acțiuni de corectare a variabilei. În primul caz, după afișarea casetei de validare, trebuie apelată funcția `Fail()` a clasei `CDataExchange` pentru a informa funcția `UpdateData()` că validarea a eșuat. Astfel, actualizarea variabilei cu conținutul controlului nu se va mai face.

Se propune implementarea unei validări asupra prenumelui, astfel: se face o validare de text, iar dacă prenumele conține și alte caractere decât litere, se împiedică transferul. Se face și o acțiune de corectare, în sensul că dacă prima literă din prenume este minusculă, ea este transformată în majusculă. Vom implementa astfel în fișierul *ModalaDlg.cpp* funcția

```

void CModalaDlg::DDV_ValidPren(CDataExchange *pDX, CString &nPren)
{
    if (pDX->m_bSaveAndValidate)
    {
        for (int i=0;i<nPren.GetLength();i++)
        {
            if (!isalpha(nPren[i]))
            {
                MessageBox(" Nu sunt acceptate caractere nealfabetice",
                    "EOARE",MB_ICONSTOP);
                pDX->Fail();
            }
        }
    }
}

```

```

        break;
    }
}
if (!isupper(nPren[0]))
{
    TCHAR x=nPren.GetAt(0);
    x-=32;
    nPren.SetAt(0,x);
}
}
}

```

Va trebui acum să declarăm această funcție în fișierul *ModalaDlg.h*. O vom declara imediat sub declarația funcției `DoDataExchange()`:

```

...
protected:
virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support
void CModalaDlg::DDV_ValidPren(CDataExchange *pDX, CString &nPren);
...

```

Acum, vom introduce linia de validare în funcția `DoDataExchange()`, imediat sub funcția `DDX_` care descrie transferul cu variabila implicată:

```

void CModalaDlg::DoDataExchange(CDataExchange* pDX)
{
    CDialog::DoDataExchange(pDX);
   //{{AFX_DATA_MAP(CModalaDlg)
    ...
    DDX_Text(pDX, IDC_PRENUME, m_strPrenume);
    DDV_ValidPren(pDX, m_strPrenume);
    DDX_Text(pDX, IDC_VIRSTA, m_nVirsta);
    DDV_MinMaxInt(pDX, m_nVirsta, 0, 18);
   //}}AFX_DATA_MAP
}

```

În final, valorile variabilelor aparținând casetei derivate, trebuie preluate și utilizate în caseta părinte. Vom adăuga următoarele variabile membru clasei **CWiz5Dlg**: `CString m_strPrenCopil`, `CString m_strSexCopil` și `CString m_nVirstaCopil`. Acestea vor prelua valorile corespunzătoare la revenirea din caseta modală. Dar acest lucru trebuie să se întâmple numai dacă în aceasta am apăsător butonul **OK**:

```

void CWiz5Dlg::OnCopii()
{
    // TODO: Add your control notification handler code here
    CModalaDlg dlgCasetaModala(this);
    int nRetCode;
    UpdateData();
    if (!m_strNume.IsEmpty())
    {
        dlgCasetaModala.m_strNume=m_strNume;
        nRetCode = dlgCasetaModala.DoModal();
    }
    else
        MessageBox("Nu avem un nume de parinte!",
            "Eroare",MB_ICONSTOP);
    if (nRetCode==IDOK)

```

```

{
    m_strPrenCopil=dlgCasetaModala.m_strPrenume;
    m_strSexCopil=dlgCasetaModala.m_strSex;
    m_nVirstaCopil=dlgCasetaModala.m_nVirsta;
}
}

```

Va trebui acum să modificăm și funcția de afișare, ca mai jos:

```

void CWiz5Dlg::OnAfiseaza()
{
    // TODO: Add your control notification handler code here
    UpdateData();
    CString strText;
    strText += m_strNume+"    " + m_strPrenume;
    strText += "\n\n Copii: \n";
    strText+= "Nume : " + m_strNume+ "\n";
    strText+= "Prenume : " + m_strPrenCopil+ "\n";
    strText+= "Sex : " + m_strSexCopil+ "\n";
    CString temp;
    temp.Format("%d",m_nVirstaCopil);
    strText+= "Virsta : " + temp+ "\n";
    MessageBox(strText,"Nume parinte",MB_ICONEXCLAMATION);
}

```

8.3 Casete de dialog nemodale

O casetă de dialog nemodală nu acaparează interfața aplicației, așa cum o face caseta de dialog modală. Ea este afișată de obicei pentru a completa funcționalitatea tipică a aplicației prin transmiterea de mesaje către aplicația apelantă. Casetele de dialog nemodale sunt folosite de obicei ca bare de instrumente neflotante, permițând utilizatorului să efectueze click pe butoanele casetei de dialog pentru a indica opțiuni privind modul de lucru sau a exprima o alegere.

O casetă de dialog nemodală folosește ca interfață o casetă de dialog obișnuită, dar pentru acest tip de casetă butoanele **OK** și **Cancel** nu mai au un rol important. Și acest tip de casete sunt încapsulate de clase derivate din **CDialog**, modul de generare a clasei asociate fiind identic cu cel prezentat în paragraful 8.1. Diferențele esențiale între casetele de dialog modale și cele nemodale se regăsesc în modul de creare și afișare și respectiv modul de închidere.

O casetă de dialog nemodală este creată (vă reamintiți de la API?) prin intermediul funcției **Create()** a clasei **CDialog**. Această funcție poate fi apelată fie în constructorul clasei ce încapsulează caseta de dialog părinte, astfel încât să se creeze o instanță a casetei nemodale o dată cu instanțierea unei casete de dialog părinte, fie într-o altă funcție, caseta nemodală fiind creată numai la apelul acestei funcții.

Funcția **Create()** primește doi parametri: primul este identificatorul machetei care se creează, iar al doilea este un pointer (implicit **NULL**) către caseta de dialog părinte. Cum caseta de dialog nemodală nu acaparează controlul, vom putea folosi acest pointer pentru a comunica cu caseta de dialog părinte. **Create()** returnează valoarea **TRUE** în cazul reușitei, respectiv **FALSE** în cazul eșecului. În cazul succesului, când **Create()** returnează valoarea **TRUE**, fereastra există dar nu este vizibilă. Pentru a o face vizibilă, este nevoie de un apel al funcției **ShowWindow()** cu parametrul **TRUE**.

Casetele de dialog modale pot fi în general încapsulate fără probleme în interiorul unei funcții, deoarece ele preiau controlul absolut al programului în momentul în care sunt lansate în execuție. Casetele nemodale, datorită faptului că nu preiau controlul

programului, vor trebui să poată fi accesate în orice punct al acestuia. Din această cauză, este bine ca să creem dinamic caseta nemodală, prin intermediul operatorului `new`, reținând adresa ei într-o variabilă pointer globală sau membră a clasei ce încapsulează caseta de dialog părinte. Caseta de dialog nemodală poate fi închisă la nevoie, distrugând-o prin intermediul operatorului `delete`. Destructorul clasei de bază, `CDialog`, se va ocupa apoi de distrugerea casetei de dialog nemodale.

Să exemplificăm aceste lucruri, completând caseta de dialog de bază ca în fig. 8.6.

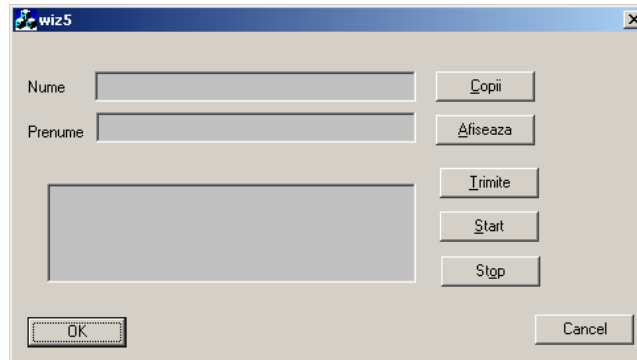


Figura 8.6. caseta de dialog părinte modificată

Am adăugat o casetă cu identificatorul `IDC_LISTA` și cu stilurile **Selection: Multiple** și respectiv **Sort** nevalidat și butoanele `IDC_TRIMITE`, `IDC_START` și `IDC_STOP`, cu etichetele respectiv **Trimite**, **Start** și **Stop**. Casetei cu listă îi asociem variabila de categorie **Control**, `CListBox m_DisplayList`.

Adăugăm apoi o resursă nouă de tip **Dialog**, căreia îi schimbăm identificatorul în `IDD_NEMODALA` și care conține controalele din fig. 8.7

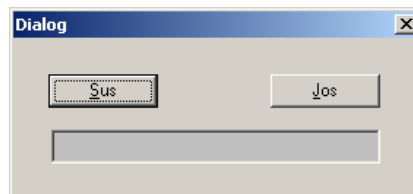


Figura 8.7. Caseta de dialog nemodală

Caseta de dialog conține butoanele cu identificatorii `IDC_SUS` și `IDC_JOS` și etichetele **sus** și respectiv **jos** și caseta de editare `IDC_MESAJ`. Apoi, casetei de dialog nou introduse i se asociază clasa `CNemodalaDlg`, respectând pașii de la începutul capitolului. Clasa va fi implementată în fișierele *NemodalaDlg.h* și *NemodalaDlg.cpp*. În final vom asocia casetei de editare `IDC_MESAJ` variabila de categorie **Value** `CString m_Mesaj`.

Următorul pas este să modificăm constructorul clasei asociate casetei de dialog nemodale, astfel încât să stabilim operarea nemodală a casetei de dialog și să o afișăm în cazul în care a putut fi creată. Se va modifica constructorul, ca mai jos:

```
CNemodalaDlg::CNemodalaDlg(CWnd* pParent /*=NULL*/)
: CDialog(CNemodalaDlg::IDD, pParent)
{
   //{{AFX_DATA_INIT(CNemodalaDlg)
    if (Create(IDD_NEMODALA,pParent))
        ShowWindow(TRUE);
    m_strMesaj = _T("");
    //}}AFX_DATA_INIT
```

```
}
```

Pentru a deschide și închide caseta de dialog nemodală, vom asocia mesajului `BN_CLICKED` generat de butoanele `IDC_START` și `IDC_STOP` funcții. Va trebui întâi să includem definiția clasei `CNemodalaDlg` în fișierul `wiz5Dlg.cpp`: Vom declara apoi un pointer global, `p_Nemodala`, care va menține adresa casetei de dialog nemodale. Dacă nu este lansată nici o casetă de dialog, acest pointer trebuie să fie `NULL`.

```
// wiz5Dlg.cpp : implementation file
```

```
...
#include "ModalaDlg.h"
#include "NemodalaDlg.h"
```

```
#ifdef _DEBUG
...
#endif
```

```
CNemodalaDlg* p_Nemodala=NULL;
```

De acum înainte, dacă există lansată o casetă nemodală, orice variabilă sau funcție din clasa asociată acesteia, va putea fi apelată din orice punct al programului, prin intermediul acestui pointer. Implementările funcțiilor de tratare a apăsării butoanelor sunt:

```
void CWiz5Dlg::OnStart()
{
    // TODO: Add your control notification handler code here
    if (!p_Nemodala)
        p_Nemodala = new CNemodalaDlg(this);
}

void CWiz5Dlg::OnStop()
{
    // TODO: Add your control notification handler code here
    if (p_Nemodala)
    {
        delete p_Nemodala;
        p_Nemodala=NULL;
    }
}
```

Funcția `OnStart()` testează dacă pointerul global are valoarea `NULL`. Dacă da, înseamnă că nu avem o casetă de dialog modală deschisă și este creată o instanță a acesteia. Să ne reamintim că în constructor era apelată funcția `ShowWindow(TRUE)`, deci caseta va fi automat afișată. Dacă nu testăm faptul că `p_Nemodala=NULL`, putem lansa oricâte casete nemodale, dar cum în programul principal există un singur pointer care menține adresa casetelor nemodale, la fiecare nouă lansare, legătura cu caseta nemodală precedentă este pierdută.

Funcția `OnStop()` testează dacă pointerul global are valoarea `NULL`. Dacă nu o are, înseamnă că avem o casetă deschisă și atunci ea poate fi distrusă, de închiderea ei ocupându-se, după cum s-a arătat anterior, destructorul clasei `CDialog`. Apoi pointerului global `i` se atribuie valoarea `NULL`, pentru a specifica că respectiva casetă nu e deschisă.

Transmiterea de valori din caseta de dialog părinte către controale sau variabile ale casetei de dialog nemodale, sau apelurile funcțiilor membru ale unei casete de dialog nemodale se pot efectua pe toată durata de viață a acesteia, prin intermediul

pointerului de acces corespunzător. La transferul de date între controale și variabilele membru mapate, prin `UpdateData()` și `DoDataExchange()` se aplică regulile deja cunoscute. Este posibil de asemenea să fie apelate funcții sau să fie fixate variabile membru ale altor obiecte ale aplicației în cadrul casetei de dialog modale, ca răspuns la interacțiunea utilizatorului cu controalele. În acest scop, trebuie transmiși casetei de dialog pointeri la obiectele respective ale aplicației. Constructorul casetei de dialog nemodale este un loc bun pentru transmiterea acestor pointeri. Pentru aceasta, se modifică constructorul clasei asociate, astfel încât la crearea casetei de dialog să fie transmiși pointeri la obiectele respective ale aplicației. Apoi acești pointeri pot fi reținuți de către caseta de dialog sub formă de variabile membru, astfel încât oricare dintre funcțiile componente va putea accesa obiectele în cauză.

Ca exemplu, vom scrie programul astfel încât la apăsarea butoanelor `IDC_SUS` și `IDC_JOS` a casetei nemodale, să fie transmise și afișate mesajele *****SUS***** și respectiv *****JOS***** în caseta cu listă. De asemenea, în sens invers, la apăsarea butonului `IDC_TRIMITE` a casetei de bază, în caseta de editare `IDC_MESAJ` a casetei nemodale va fi afișat mesajul *“Am primit mesajul **SUS**”* sau *“Am primit mesajul **JOS**”* în funcție de ultimul mesaj primit, sau *“Nu am primit nimic”* dacă nu a fost transmis nici un mesaj.

Prima dată, va trebui să modificăm constructorul clasei casetei de dialog nemodale, pentru a ne asigura că pointerul transmis către părinte este `CWiz5Dlg*` și nu pointerul `CWnd*` curent. Pentru aceasta, se ajustează ca mai jos definiția constructorului din fișierul antet *NemodalaDlg.h*:

```
...
class CNemodalaDlg : public CDialog
{
// Construction
public:
    CNemodalaDlg(CWiz5Dlg* pParent = NULL);
...

```

De asemenea, la începutul fișierului cu definiția clasei `CNemodalaDlg`, adică *NemodalaDlg.cpp*, va trebui să inserăm fișierul de definiție al clasei `CWiz5Dlg`. Acest fișier trebuie inclus înaintea includerii fișierului *NemodalaDlg.h*, pentru că în acesta apare o referire la clasa `CWiz5Dlg`.

```
// NemodalaDlg.cpp : implementation file
#include "stdafx.h"
#include "wiz5.h"
#include "wiz5Dlg.h"
#include "NemodalaDlg.h"

```

Pointerul `pParent` spre caseta de dialog părinte există doar în constructorul clasei casetei de dialog nemodale. Noi va trebui să-l memorăm, astfel încât să-l putem utiliza în diferite funcții ale clasei. Pentru aceea, vom adăuga clasei `CNemodalaDlg` variabila membru ***protejată*** `CWiz5Dlg* m_pParent`. Acesta va fi pointerul în care se va copia `pParent`. Vom modifica constructorul clasei care încapsulează caseta de dialog nemodală, ca mai jos:

```
CNemodalaDlg::CNemodalaDlg(CWiz5Dlg* pParent /*=NULL*/)
: CDialog(CNemodalaDlg::IDD, pParent)

```

```

{
    //{AFX_DATA_INIT(CNemodalaDlg)
    if (Create(IDD_NEMODALA,pParent))
    {
        ShowWindow(TRUE);
        m_pParent=pParent;
    }
    m_strMesaj = _T("");
    //}}AFX_DATA_INIT
}

```

Acum, putem asocia funcțiile pentru apăsarea butoanelor din caseta de dialog nemodală, accesând elemente din caseta de dialog părinte prin intermediul pointerului `m_pParent`:

```

void CNemodalaDlg::OnSus()
{
    // TODO: Add your control notification handler code here
    m_pParent->m_DisplayList.AddString("*** SUS ***");
}

void CNemodalaDlg::OnJos()
{
    // TODO: Add your control notification handler code here
    m_pParent->m_DisplayList.AddString("*** JOS ***");
}

```

De asemenea, putem implementa funcția ce răspunde la evenimentul `BN_CLICKED` corespunzător butonului `IDC_TRIMITE` din caseta de dialog părinte.

```

void CWiz5Dlg::OnTrimite()
{
    // TODO: Add your control notification handler code here
    int nr;
    CString Mesaj;
    nr=m_DisplayList.GetCount();
    if (p_Nemodala) {
        if (nr)
        {
            m_DisplayList.GetText(nr-1, Mesaj);
            p_Nemodala->m_strMesaj = "Am primit mesajul " + Mesaj;
        }
        else p_Nemodala->m_strMesaj = CString("Nu am primit nimic!");
        p_Nemodala->UpdateData(FALSE);
    }
}

```

Se observă ca accesul la caseta de editare din caseta de dialog nemodală este făcut tot pe baza pointerului la caseta de dialog respectivă.

În acest moment, programul este funcțional, dar are o mică problemă. Cu toate că din macheta casetei nemodale au fost înlăturate butoanele **OK** și **Cancel**, aceasta poate fi închisă forțat prin click pe pictograma cruciuliță din dreapta sus. Prin închiderea forțată a casetei, dacă fereastra principală nu interceptează un mesaj de închidere, pointerul și memoria asociate pentru caseta nemodală rămân alocate, cu toate că obiectul nu mai există. Aceasta poate duce la pierdere de memorie și în plus, nu va putea fi apelată o nouă instanță, până când vechea instanță nu va fi închisă corect.

Rezolvarea acestei probleme se poate face în două moduri:

- interceptarea mesajului `WM_CLOSE` pentru clasa `CNemodalDlg`. Să ne reamintim cum: în **ClassWizard**, la pagina **Message Maps**, în caseta combinată **Class name**: se alege `CNemodalDlg`, în caseta **Object IDs**: tot `CNemodalDlg`, iar în caseta cu listă **Messages**: se alege `WM_CLOSE`. Se apasă apoi butoanele **Add Function** și **Edit Code**.

```
extern CNemodalDlg* p_Nemodala;
void CNemodalDlg::OnClose()
{
    // TODO: Add your message handler code here and/or call default
    CDialog::OnClose();
    delete(this);
    p_Nemodala=NULL;
}
```

Definirea pointerului global extern `p_Nemodala` ne asigură faptul că acesta este același pointer cu cel definit în clasa `CWiz5Dlg`. În funcție, după distrugerea casetei de dialog nemodale, acest pointer este făcut `NULL`.

- înlăturarea butonului cruciuliță, prin deselectarea stilului **System menu** pentru caseta nemodală.

Întrebări și probleme propuse

1. Implementați și executați toate exemplele propuse în capitolul 8;
2. Adăugați proiectului o nouă casetă de dialog, pe care să o lansați nemodal din caseta de dialog modală. Ce se întâmplă dacă închidem caseta de dialog modală în timp ce caseta de dialog nemodală copil este deschisă? Explicați.
3. Adăugați mai multe casete de dialog proiectului, pe care să le lansați nemodal în cascadă. Ce se întâmplă dacă se închid casete de la baza lanțului? Explicați.