

3. Laborator

Lucru cu registrii SO Windows Embedded Compact

3.1. Introducere

Scopul laboratorului de astăzi este de a înțelege modalitate prin intermediul căreia putem citi, scrie sau șterge informații din baza de date *registry*.

Baza de date *registry* are o organizare ierarhică, structurată sub forma unui arbore, și conține o serie largă de informații (de multe ori critice) necesare sistemului de operare Windows (diferitelor componente ale sale, precum *device manager*-ului sau unui server de FTP), serviciilor și aplicațiilor ce rulează în cadrul SO Windows. Fiecare nod existent în structura arborescentă a bazei de date *registry* poartă numele de cheie. În cadrul SO Windows Embedded Compact această structură arborescentă are maxim 16 nivele de subchei. Numărul maxim de caractere pe care o cheie o poate avea este de 255 fără caracterul nul (valoare hexa 0x00), cel de terminare a șirului. Fiecare cheie poate conține alte subchei sau date de diferite tipuri. Aceste date poartă numele de valori. Numărul de subchei și/sau valori, conținute de o anumită cheie, nu este limitat.



Figura 3.1. Interfața grafică cu utilizatorul a programului ce trebuie dezvoltat

3.2. Cerințe

Creați un program care să aibă o interfață grafică similară cu cea din **Figura 3.1**. Programul va avea următoarea funcționalitate:

1. La apăsarea butonului “Scrie reg.” se vor prelua informațiile **Nume** (câmp ce poate conține orice înșiruire de caractere) și **Vârsta** (o valoare întreagă) și vor fi salvate într-o nouă cheie, care va fi creată sub numele “STUDENT” în cadrul secțiunii **HKEY_LOCAL_MACHINE** a regiștrilor sistemului de operare. În cadrul acestei noi chei vor exista câmpurile “Varsta” și “Nume” unde se vor salva informațiile preluate

de pe interfața grafică. Programul va verifica și existența unor informații corecte în cadrul acestor câmpuri. Doar dacă informațiile sunt valide se va trece la pasul imediat următor.

2. La apăsarea butonului “Citește reg.” se vor citi din cheia “STUDENT” informațiile de nume și vârstă și se vor afișa în câmpurile asociate de pe interfața grafică.
3. La apăsare butonului “Sterge reg.” se șterge cheia “STUDENT” cu ambele câmpuri existente în interiorul ei.
4. După executarea fiecărei funcții asupra bazei de date registry se va afișa dacă s-a realizat cu succes sau nu, această funcție, prin intermediul unor ferestre de mesaje, altele și independente de fereastra grafică principală a programului.
5. La apăsarea butonului “Șterge msg. Edit Controls” informația existentă în cele două elemente Edit Control va fi ștearsă.

3.3. Prezentarea funcțiilor de bază utilizate în manipularea registrilor

Pentru citirea, scrierea sau ștergerea diferitelor chei sau valori din acestea se folosesc o serie de funcții a căror scurtă descriere este prezentată în cele ce urmează. Prezentarea funcțiilor este una succintă, utilizați *help*-ul mediului Visual Studio pentru înțelegerea rolului și a funcționalităților generate de utilizarea diferitelor argumente posibile ale acestor funcții.

Scurtă descriere a funcțiilor care manipulează regiștri:

1. Pentru crearea sau deschiderea unei noi chei folosiți funcția: [RegCreateKeyEx \(arg1, ..., arg8, arg9\)](#).

În cadrul acestei funcții câmpul **arg1** poate fi un handle către o anumită cheie (obținut cu una din funcțiile [RegCreateKeyEx](#) sau [RegOpenKeyEx](#)) sau numele unei chei predefinite, atunci când se dorește crearea altei chei în aceasta. Din cheile predefinite mentionam:

- a. **HKEY_LOCAL_MACHINE** – conține informații legate de sistemul pe care este instalat SO, informații ce țin de partea *hardware* și *software* a SO – memorie, drivere existente, partea de initializare a SO (programe ce se pornesc, securitate etc)
- b. **HKEY_CURRENT_USER** – conține informații despre profilul utilizatorului logat în acel moment în sistem (variabile sistem, setări desktop, preferințe aplicații, organizare programe personale în Start meniu etc.). Este creat de fiecare dată când utilizatorul se loghează.
- c. **HKEY_CLASSES_ROOT**
- d. **HKEY_CURRENT_CONFIG**
- e. **HKEY_USERS**

Pentru **arg4** trimiteți funcției NULL.

Un alt câmp important este **arg8**, acesta este adresa unei variabile de tip HKEY în care se va stoca un identificator unic pentru cheia nou creată care va fi utilizat ulterior în cadrul tuturor funcțiilor ce fac referire la această cheie.

În situația creării cu succes a noii chei **arg9** este egal cu **REG_CREATED_NEW_KEY**.

2. [RegOpenKeyEx](#) – citire chei și întoarcere *handler* de identificare unic al acelei chei. Trebuie utilizat înaintea unei operații de citire de valori dacă nu avem disponibil un *handler* către o astfel de cheie obținut anterior printr-o funcție de tipul [RegCreateKeyEx](#).
3. [RegCloseKey](#) funcție ce trebuie utilizată la final (după finalizarea tuturor operațiilor de manipulare a datelor din regiștri), după ce am utilizat una din funcțiile [RegCreateKeyEx](#) sau [RegOpenKeyEx](#).
4. [RegSetValueEx](#) (*arg1*, ..., *arg4*, ...) – funcție utilizată în crearea diferitelor valori de tip text ([REG_SZ](#) pentru *arg4*) sau numerice (de ex. [REG_DWORD](#) pentru *arg4*). *Arg1* va fi identificatorul unic de tip [HKEY](#) obținut cu ajutorul funcției [RegCreateKeyEx](#) sau [RegOpenKeyEx](#).
5. [RegQueryValueEx](#) – citire valori din regiștri.

În momentul în care citim din regiștri "Varsta" deoarece tipul de dată este [DWORD](#) ne vine foarte ușor să trimitem ultimul argument al funcției (care ne cere lungimea datelor ce vor fi preluate din regiștri). Codul este următorul:

```
dwCati = sizeof(DWORD);
err = RegQueryValueEx(hKey, L"Varsta", NULL, &dwType,
                    (LPBYTE) &dwVarsta, &dwCati);
```

În cazul valorii "Nume" din cheia "STUDENT" deoarece șirul de caractere poate avea o lungime variabilă prima dată se apelează funcția [RegQueryValueEx](#) pentru a lua lungimea șirului de caractere și ulterior pentru a lua efectiv șirul de caractere. Codul ce poate fi utilizat pentru aceasta este:

```
TCHAR nume[100];
// preiau lungimea șirului de caractere in variabila dwCati
err = RegQueryValueEx(hKey, L"Nume", NULL, &dwType, NULL, &dwCati);
// preiau numele, șirul de caractere in variabila nume
err = RegQueryValueEx(hKey, L"Nume", NULL, &dwType,
                    (LPBYTE) nume, &dwCati);
```

6. [RegDeleteKey](#) - ștergere cheie cu toate valorile existente în ea.

3.4. Informații suplimentare necesare îndeplinirii obiectivelor

Pentru preluarea unei informații numerice dintr-un `EditBox` parcurgeți urătorii pași:

1. Asociați elementului de tip `Edit Control` o variabilă de tip `control` (nu de tip `valoare`) – conform prezentării care s-a făcut în **Laboratorul 2**.
2. În urma pașilor anteriori în cadrul clasei ce gestionează interfața grafică veți observa apariția unei declarații de variabilă (clasă în cazul nostru) similară cu:

```
CEdit mVarsta;
```

3. În interiorul funcției care va realiza preluarea informației de pe interfața grafică înserați următoarea declarație:

```
CString szVarsta;
```

4. Cu ajutorul funcției `GetWindowTextW` puteți prelua informația din elementul `Edit Control` și depozita în `szVarsta`:

```
mVarsta.GetWindowTextW(szVarsta);
```

5. În acest moment datele numerice din `EditText` sunt stocate local într-o structură de tip `CString`.
6. Pentru obținerea valorii numerice va trebui să utilizați funcția `wcstoul` (“wcs to ul” funcție ce convertește un element de tip **CString** către **(to)** un element de tip *unsigned long* (**ul – unsigned long**)):

```
DWORD varsta;  
varsta = wcstoul(szVarsta.GetBuffer(), '\0', 10);
```

Primul argument al funcției este *buffer*-ul de tip text care conține valoarea numerică, al doilea argument definește cu ce caracter se termina textul, în cazul nostru cu o valoare 0, ultimul argument este baza în care se lucrează.

S-a ales o conversie către un element de tip `DWORD` deoarece funcția care va salva valoarea în regiștri (`RegSetValueEx`) accepta argumente numerice de acest tip.

7. Se scrie valoarea în regiștri:

```
err = RegSetValueEx(hKey, _T("Varsta"), 0, REG_DWORD,  
                    (LPBYTE)&varsta, sizeof(varsta));
```

Pentru salvarea unei informații text dintr-un element de tip `Edit Box` într-un câmp în cadrul unei chei aflată în regiștri SO executați următorii pași:

1. Parcurgeți în mod similar pașii de la 1 la 5 prezentați anterior – similar doar din punct de vedere conceptual, în sensul că exista mici diferențe precum: variabila de tip control, nu va mai fi “`mVarsta`”, înlocuiți-o cu alta de exemplu “`mNume`” și asociați-o corect cu elementul de pe interfață, variabila “`szVarsta`” o înlocuiți cu alta (de exemplu “`szNume`”) etc.
2. Pentru salvarea textului în regiștri utilizați:

```
err = RegSetValueEx(..., ..., 0, REG_SZ, (LPBYTE)szNume.GetBuffer(),  
                    (DWORD)(lstrlen(szNume.GetBuffer()) + 1)*sizeof(TCHAR));
```

Se observă preluarea *buffer*-ului de date prin intermediul `szNume.GetBuffer()` din cadrul structurii `CString`.

3. Pentru afișarea diferitelor mesaje (de eroare sau de finalizare cu succes a funcției) este indicat ca să se preia (într-o variabilă - în cazul nostru variabila `err`) și să se trateze posibilele mesaje de eroare, având grijă că dacă acestea există programul să țină cont de

ele (de ex. cheia STUDENT nu a putut fi creată, atunci nu se trece la scrierea valorilor preluate de pe interfața grafică). Pentru afișarea finalizării cu succes sau nu aCeva de genu:

```
if (err == ERROR_SUCCESS)
    AfxMessageBox(_T("S-a scris nume!!!!"), MB_OK, 0);
else
    AfxMessageBox(_T("Nu s-a scris nume!!!!"), MB_OK, 0);
```

Funcțiile `AfxMessageBox` de mai sus, afișează într-un box un mesaj, blochează interfața cu utilizatorul și rularea programului până în momentul în care acesta apasă butonul OK, fereastra nou deschisă se închide și programul își continuă execuția trecând de linia cu codul `AfxMessageBox(_T ...` la executarea următoarei.

Cel de al doilea argument poate preciza și numărul și tipul butoanelor existente pe fereastra de dialog nou deschisă. Câteva constante ce se pot preciza sunt:

- **MB_ABORTRETRYIGNORE** – fereastra de mesaje va conține butoanele: Abort, Retry și Ignore.
- **MB_OK** – fereastra de mesaje va conține butonul: OK.
- **MB_OKCANCEL** – fereastra de mesaje va conține butoanele: OK și Cancel.
- **MB_RETRYCANCEL** – fereastra de mesaje va conține butoanele: Retry și Cancel.
- **MB_YESNO** – fereastra de mesaje va conține butoanele: Yes și No.
- **MB_YESNOCANCEL** – fereastra de mesaje va conține butoanele: Yes, No și Cancel.

Pentru alte informații despre această funcție apelați la help-ul mediului Visual Studio.

De asemenea înaintea interogării elementelor de tip Edit Control puteți verifica dacă există date în ele:

```
if (mNume.GetWindowTextLengthW() == 0)
{
    AfxMessageBox(_T("Informatie nume inexistentă !"), MB_OK, 0);
    return; //se termină funcția deoarece un TextBox este gol
}
```

În cazul în care funcția `AfxMessageBox` nu funcționează utilizați funcția `OutputDebugString` cu deavantajul că de această dată mesajele vor fi afișate în fereastra de Debug a mediului Visual Studio și nu drept o fereastră de tip mesaj pe placa de dezvoltare.

```
OutputDebugString(_T("orice text doriți dvs. \n"));
```

3.5. Exerciții

1. Realizați verificarea faptului că în câmpul “Varsta” este un număr întreg.

2. În cheia “STUDENT” creați o subcheie “Date_Importante” în care să stocați două valori: data de naștere și data admiterii la facultate. Dezvoltați toate funcționalitățile programului astfel încât să poată scrie, citi și șterge și aceste câmpuri (evident aceasta înseamnă și modificările asociate interfeței grafice).