Programarea aplicațiilor Microsoft Office (Visual Basic for Applications)

Introducere

VBA (Visual Basic for Applications) este un limbaj de programare creat de Microsoft pentru automatizarea aplicațiilor. Ataşat inițial componentelor din Microsoft Office, în prezent sunt raportate peste 200 de aplicații care include VBA (CorelDraw, AutoCAD etc.).

VBA este parte a familiei de limbaje Visual Basic, situându-se sub VB, dar peste VBScript. Totuşi, VBA este **acum** un element esențial în versiunea completă VB, oferind suport pentru limbaj, interfața pentru forme, controale, obiecte, tehnologii de accesare a datelor.

Atunci când este găzduită de altă aplicație, cum ar fi Excel, VBA oferă mijloacele de interacțiune cu obiectele aplicației gazdă. In acest caz, VBA permite dezvoltatorilor să furnizeze soluții complete care extind şi/sau integrează aplicațiile gazdă.

Pentru a programa în VBA trebuie totuşi reținut că o cerință supli-mentară față de alte limbaje este aceea că trebuie să se cunoască aplicația gazdă (Word, Excel, PowerPoint, Access etc.).

Scurt istoric

1993 — VBA apare în Microsoft Excel

1994 — VBA este ataşat la Microsoft Project

1995 — este inclus în Microsoft Access, înlocuind Access Basic

1996 — VBA devine element în Visual Basic

1996 — este inclus în Word, înlocuind Word Basic

1997—VBA este integrat în suita Office 97

1997 — Microsoft licențiază VBA pentru utilizarea în alte aplicații software

Editorul Visual Basic

În această secțiune se prezintă mediul de dezvoltare Visual Basic for Applications integrat în Microsoft Office.

Utilizând Visual Basic Editor, numit în continuare VBE, se poate crea, edita, depana şi executa cod program asociat cu documente Microsoft Office.

Proiectele dezvoltate în VBE, deşi sunt asociate aplicațiilor din Office, nu pot fi reduse, ca problematică, la procesarea de texte (Word), calcul tabelar (Excel), prezentări electronice (PowerPoint) sau baze de date (Access). Este corect să se considere aceste proiecte drept aplicații similare celor dezvoltate în alte medii de programare, având însă la dispoziție componentele aplicațiilor din Office. Cu alte cuvinte, nu este vorba de o limitare a posibilităților de prelucrare, ci o potențare a acestora prin apelul posibil la obiectele din Office.

O obiecție la utilizarea VBA este aceea că proiectul se poate executa doar dintr-o aplicație Office (deci deschizând, chiar formal, un document Word, sau o foaie Excel etc.), dar multitudinea de componente disponibile în dezvoltarea proiectului compensează acest neajuns. În plus nu trebuie uitat că orice aplicație necesită o interfață utilizator (puternică în Microsoft Office) şi că aplicațiile de bază sunt întreținute şi completate de Microsoft, astfel încât proiectele noastre se vor actualiza şi ele o dată cu componentele Office.

Un ultim argument este acela că mediul VBE este identic cu mediul de dezvoltare din Microsoft Visual Studio (Visual Basic, C++ etc.) astfel că practica în VBA poate fi considerată introductivă către alte aplicații RAD.

Interfața grafică VBE

Pentru a deschide editorul VB, mai întâi se va porni o aplicație din Microsoft Office, apoi se poate acționa

combinația Alt+F11 (dacă nu a fost atribuită altei operațiuni), sau

butonul Visual Basic Editor de pe bara de unelte Visual Basic (meniul View, Toolbars etc.) vizualizată într-o aplicație Office, sau

Meniul Tools, Macro, Visual Basic Editor.

Interfața grafică VBE este suficient de complexă, asemănătoare mediilor de programare din Visual Studio. Pe lângă obiectele grafice uzuale (Menu Bar, bare de unelte) sunt disponibile ferestre specializate pentru lucrul cu anumite categorii de obiecte:

Properties Window pentru vizualizarea şi fixarea proprietăților în momentul proiectării (design-time);

Project Explorer care permite navigarea, vizualizarea şi modificarea proiectelor deschise la un moment dat;

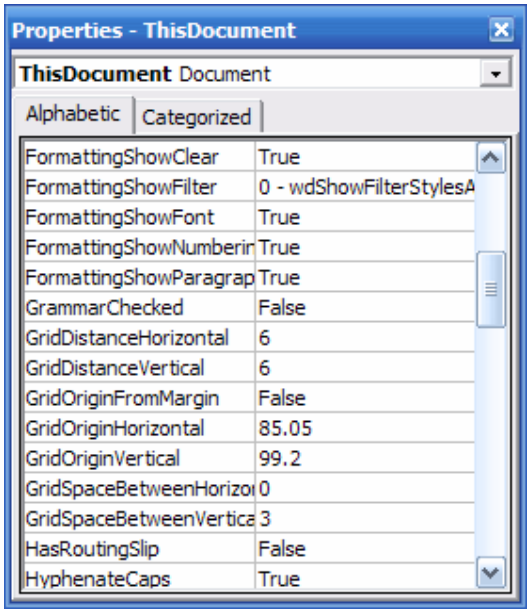
Code Window unde este se scrie şi este vizibil codul sursă al proiectului activ;

Locals Window care permite vizualizarea variabilelor locale cu valorile lor;

Immediate Window care permite executarea imediată a unei linii de cod;

Watch Window unde se afişează valorile unor expresii specificate (utile în depanarea programelor).

Properties Window



Prin **proprietate** a unui obiect se înțelege o caracteristică a respectivului obiect (cum ar fi culoarea sau vizibilitatea etc.). Fixarea valorii proprietății se reflectă în aparența obiectului sau în comportamentul lui (de exemplu, fixarea proprietății ShowSpellingErrors la valoarea True arată în document erorile de scriere).

Fereastra Properties poate fi utilizată, în momentul proiectării, pentru a vizualiza toate proprietățile obiectului activ şi a modifica valorile dorite.

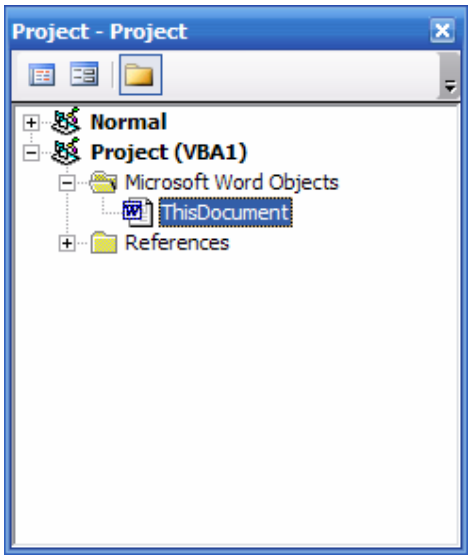
În partea superioară este cutia de obiecte în care se poate selecta un obiect (sau mai multe) dintre cele vizibile.

În fişa Alphabetic se listează proprietățile modificabile ale obiectului selectat, în ordine alfabetică. Se poate modifica valoarea unui atribut prin selectarea numelui proprietății şi tastarea sau selectarea noii valori.

În fişa Categorized sunt listate proprietățile după categorii, într-un control de tip Explorer, in care ramurile pot fi expandate sau.

Fereastra Properties poate fi arătată (când nu este vizibilă), prin comanda **Properties Window** din meniul **View**.

Project Explorer



Codul sursă asociat cu un workbook, document, template sau prezentare este asociat într-un **proiect,** care este memorat şi salvat în mod automat o dată cu caietul Excel, documentul Word, şablonul sau prezentarea respectivă. În fereastra Project Explorer se pot vedea, modifica şi naviga printre toate proiectele asociate oricărui document, caiet, şablon sau prezentare deschise.

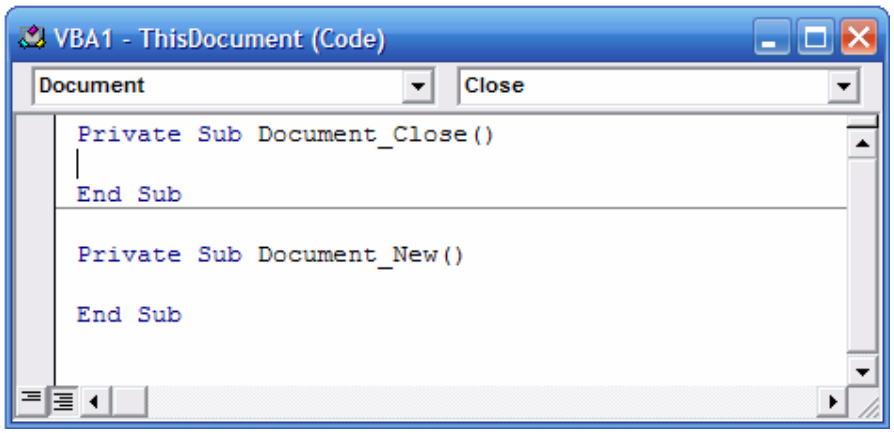
Pentru un proiect se listează, într-un control de tip Explorer, obiectele care recunosc evenimente, formele, modulele, referințele. Pentru a vedea codul dintr-un modul sau codul asociat unui obiect, se selectează respectivul modul sau obiect şi se apasă butonul View Code (primul din stânga). Pentru a vedea interfața utilizator pentru un obiect sau formă se selectează şi se apasă butonul View Object (cel din mijloc). Pentru a vedea organizarea în foldere a elementelor listate în Project Explorer se va apăsa butonul Toggle Folders.

Fiecare element este însoțit, în arborele de structură, de icoana specifică elementului: proiect, folder, referință, obiect etc.

Code Window

Fereastra principală a Editorului Visual Basic este cea în care se poate scrie codul sursă. Deoarece procedurile sunt asociate unor obiecte de interfață, sau aparțin unui modul, mai întâi se va selecta, din Project Explorer, modulul sau forma vizată şi apoi se va apăsa butonul View Code.

Pentru a vedea mai mult de o procedură în fereastra de cod, se va selecta boxa de control **Default to Full Module View** în fişa **Editor** din **Options** (meniul **Tools**) (în caz contrar se va vedea doar câte o procedură).



În partea de sus a ferestrei se găsesc două boxe:

Object Box unde se poate selecta obiectul pentru care se afişează procedurile asociate;

Procedures/Events Box unde se poate selecta procedura pentru care se vizualizează/editează codul. Procedurile pot fi de tip eveniment, dacă obiectul selectat este o formă utilizator. Selectarea unei proceduri produce o defilare a textului astfel încât pointerul să fie la prima linie a procedurii.

Pot fi deschise mai multe ferestre de editare, textul poate fi mutat/copiat între ferestre, ferestrele se pot diviza utilizând bara de divizare etc.

Acționarea butonului Procedure View Icon (primul din stânga, pe bara de jos a ferestrei) sau a butonului Full Module View Icon produce alegerea între vizualizarea unei singure proceduri sau a tuturor procedurilor din modul.

Locals Window

Utilă în procesul de punere la punct a programului, fereastra Locals serveşte la afişarea automată a tuturor variabilelor declarate în procedura curentă. Conținutul ferestrei este actualizat de fiecare dată când se trece din modul Run în modul Break sau atunci când se navighează în stiva de apeluri.

Pentru o descriere a ferestrei se va vedea secțiunea dedicată depanării programelor.

Immediate Window

Permite scrierea şi execuția imediată a unei linii de cod. Linia poate fi copiată în şi dintr-o fereastră de cod.

În modul de execuție Break, instrucțiunea din fereastra Immediate este executată în contextul procedurii afişate în Procedure Box.

Pentru acțiunile posibile în fereastra Immediate, se va studia Help – Immediate Window Keyboard Shortcuts.

Watch Window

Este fereastra unse sunt afişate automat valorile expresiilor urmărite în etapa de depanare a proiectului.

Pentru o descriere a ferestrei Watch se va vedea secțiunea dedicată depanării programelor.

Scrierea procedurilor

Instrucțiunile unui proiect se pot înscrie, după modul lor de creare, în două mari categorii:

scrise de aplicația de bază (Word, Excel etc.) prin traducerea acțiunilor interfeței utilizator (meniuri, comenzi etc.) în cod VBA. Această operațiune este cea de înregistrare a unui macro.

scrise într-o fereastră de cod de către utilizator (proiectant), cu asistența mediului VBE.

Înregistrarea unui macro

Acțiunea este utilă atât prin aceea că operațiuni simple pot fi traduse uşor în instrucțiuni VBA, procedurile pot fi editate din VBE, iar pentru proceduri mai complexe (cum ar fi operatiuni de căutare/înlocuire sau formatări de

obiecte grafice) codul generat automat oferă un bun model de utilizare a obiectelor, proprietăților şi metodelor aplicației.

Pentru a înregistra un macro:

- Se afişează bara de unelte **Visual Basic** (meniul **View** - **Toolbars** şi selectarea barei dorite).
- Pe bara **Visual Basic** se acționează butonul **Record Macro**.
- În dialogul **Record Macro** se înlocuieşte numele dat implicit în boxa **Macro Name** şi apoi **OK**.

  Se poate utiliza boxa **Store macro** pentru a alege locul de memorare a codului.

Se execută acțiunile menite să fie înregistrate/traduse în VB, în succesiunea dorită.

- Pe bara **Stop Recording** (apărută la inițierea procesului de înregistrare) se apasă butonul **Stop Recording**.

Pentru a vedea liniile de cod generate, se deschide în aplicația de bază meniul **Tools**, comanda **Macro**, apoi **Macros**, se selectează după denumire şi se apasă butonul **Edit**.

Codul sursă poate fi văzut şi prin navigarea în VBE prin **Project Explorer**, ferestre de cod etc.

Din punctul de vedere al programării se poate spune că un macro este o procedură publică fără argumente, deci poate fi scrisă şi direct în fereastra de cod a unui document. Din punct de vedere formal, toate procedurile care pot fi executate din dialogul Macros (**Tools** - **Macro** - **Macros**) sunt macro-uri.

Scrierea unei proceduri

Dacă se doreşte scrierea unor proceduri generale, care nu sunt asociate unui obiect sau eveniment specific, se ca crea o procedură într-un modul standard.

Pentru a crea un modul standard nou (gol), se merge în **Project Explorer** în proiectul unde se adaugă modulul nou creat şi se dă comanda **Module** din meniul **Insert**.

Pentru a deschide un modul standard existent, se va selecta modulul din **Project Explorer** şi se apasă butonul **View Code** (sau dublu click).

Pentru a adăuga o procedură la un modul, se selectează modulul în Project Explorer, se deschide meniul Insert şi se dă comanda Procedure. Se deschide dialogul Add Procedure unde se vor selecta opțiunile definitorii (subrutină sau funcție, publică sau nu etc.) şi se dă OK. După aceasta se pot adăuga liniile de cod ale procedurii.

Scrierea unei proceduri de eveniment (event procedure)

Dacă se doreşte scrierea de cod sursă care să se execute automat atunci când are loc un anumit eveniment (cum ar fi deschiderea unui document, acționarea unui buton etc.), trebuie să se scrie o procedură asociată evenimentului respectiv. O asemenea procedură se va numi procedura evenimentului.

Anumite obiecte din aplicațiile Microsoft Office recunosc un set predefinit de evenimente, care pot fi declanşate de către sistem sau de către utilizator. Evenimentele specifice fiecărui obiect trebuie să fie studiate separat (se va studia secțiunea din Help pentru fiecare aplicație), doa principalele obiecte, cu proprietățile, metodele si evenimentele lor, sunt prezentate şi în acest curs, în capitole separate.

Modul cum aplicația răspunde la evenimentele recunoscute poate fi controlat prin scrierea procedurilor de eveniment. O asemenea procedură se va scrie în fereastra Code asociată obiectului. De fiecare dată când apare evenimentul se execută procedura evenimentului respectiv. De exemplu, dacă se scrie o procedură asociată cu evenimentul Open al unui document Word, procedura se va executa automat la fiecare deschidere a acelui documentului.

O procedură de eveniment este memorată în documentul, caietul, foaia de calcul, diapozitivul, forma utilizator etc. unde poate fi declanşat evenimentul. Pentru a vedea codul sursă al procedurii, se va selecta obiectul în Project Explorer şi click pe butonul View Code pentru a deschide fereastra de cod asociată. Dintr-o fereastră de cod deschisă, asociată, se va selecta obiectul vizat, din boxa de obiecte, şi în boxa de proceduri vor fi listate atunci toate procedurile evenimentelor, chiar dacă ele nu sunt efectiv scrise. Selectarea unui eveniment va scrie (dacă nu există) liniile obligatorii ale procedurii şi va fixa cursorul de editare în procedura respectivă.

Numele unei proceduri de eveniment este format din numele obiectului, care recunoaşte evenimentul, urmat de caracterul "_" şi de numele evenimentului asociat. De exemplu, Document_Open este numele procedurii care se execută la deschiderea unui document.

Pentru controale ActiveX, numele este legat de numele codului controlului. Schimbarea numelui codului după ce s-au scris procedurile evenimentelor impune modificarea denumirilor acestora. La cele mai multe obiecte (Document, Worksheet, UserForm) denumirile sunt legate de numele clasei, deci nu mai trebuiesc redenumite.

Observaţie. Dacă se doreşte ca o procedură să fie asociată cu un document specific, dar nu cu un eveniment specific, atunci procedura se va scrie în secţiunea (General) a documentului respectiv (de exemplu o rutină care să poată fi apelată din mai multe proceduri de eveniment).

Unelte VBE pentru scrierea instrucţiunilor

Deoarece multe dintre denumirile obiectelor, proprietăţilor sau metodelor care apar în codul VBA sunt complexe, mediul de dezvoltare oferă o serie de unelte pentru completarea automată a cuvintelor cheie, pentru oferirea de ajutor în reamintirea denumirilor etc.

Dacă s-au tastat suficient de multe caractere încât VB poate recunoaşte un cuvânt, atunci prin CTRL+SPACE, sau click pe butonul **Complete Word** de pe bara de unelte **Edit,** completează cuvântul.

În dialogul Options (meniul Tools) se pot activa următoarele acţiuni, executate automat la completarea unei linii de cod:

verificarea automată a sintaxei — Auto Syntax Check;

obligativitatea declarării tuturor variabilelor, adăugarea automată a instrucţiunii Option Explicit la orice nou modul — Require Variable Declaration;

afişarea unei liste cu informaţii utile (logice la poziţia curentă a cursorului) la completarea instrucţiunii — Auto List Member;

afişarea informaţiei despre proceduri şi parametrii lor — Auto Quick Info;

afişează, doar în modul Break, valoarea unei variabile peste care este plasat cursorul — Auto Data Tips;

alinierea automată a liniilor noi la începutul liniei precedente — Auto Indent;

fixarea lăţimii între poziţiile tabulatorului, 1 la 32 de spaţii (implicit fiind 4) —Tab Width.

Pe bara de unelte Edit există câteva butoane, care ajută la completarea cuvintelor şi expresiilor în timpul scrierii instrucţiunilor:

List Properties/Methods — deschide o cutie în fereastra Code cu proprietăţile şi metodele permise pentru obiectul care precede caracterul punct ("."), utilă atunci când se operează cu obiecte.

List Constants — deschide în fereastra de cod, la punctul de inserţie, o cutie cu constantele permise pentru proprietatea care precede semnul egal ("=") în instrucţiunea curentă.

Quick Info — oferă, ca ajutor, sintaxa pentru o variabilă, funcţie etc. prin analiza locului punctului de inserţie pe linia curentă.

Parameter Info — arată o cutie, la punctul de inserţie, cu informaţia despre parametrii funcţiei în care este pointerul.

Complete Word — acceptă caracterele pa care le propune VBE drept completare la cuvântul tastat.

Comment Block — care transformă în comentarii liniile selectate.

Uncomment Block — înlătură semnul de comentarii la liniile selectate.

Executarea unei proceduri Sub

O procedură poate să se execute:

automat, ca răspuns la declanşarea unui eveniment (procedura evenimentului);

- din VBE, dacă punctul de inserţie este în procedură şi se acţionează butonul **Run Sub/UserForm** de pe bara de unelte **Standard** sau **Debug**;
- ca un macro, **Run** din dialogul **Macros** (**Tools** - **Macro**) al aplicaţiei de bază;

apelată din altă procedură.

La apelul unei proceduri din altă procedură se va ține seama de interacțiunea declarațiilor **Public**, **Private**, ca şi de referințele la alte proiecte (meniul **Tools** - **References**).

Instrucțiunile VBA

Tipuri de date

Variabilele şi constantele utilizate într-un program VBA pot avea diverse tipuri, specifice datelor memorate. Spre deosebire de alte limbaje de programare, există un tip universal — tipul Variant —, care poate conține aproape orice alt tip de date. Acest tip este asignat în mod implicit tuturor variabilelor nedeclarate altfel, încât declararea explicită poate fi utilizată atunci când se doreşte economisirea memoriei (tipul Variant alocă mai multă memorie), viteză în execuție sau atunci când se scriu date într-un fişier în acces direct.

Boolean

Domeniu de valori: True sau False (valorile logice)

Memorie: 2 bytes

Declarator de tip:

Observații. Convertirea valorilor numerice la tipul Boolean: 0 produce False, valorile nenule produc True.
Convertirea valorilor de tip Boolean la alte tipuri numerice: False devine 0, True devine -1.

Byte

Domeniu de valori: 0–255 (numere întregi, fără semn)

Memorie: 1 byte

Declarator de tip:

Observații.

Currency

Domeniu de valori: -922 337 203 685 477.5808 — 922 337 203 685 477.5807

Memorie: 8 bytes

Declarator de tip: @

Observații. Utilizate pentru calcule băneşti (sau alte situații în care precizia este foarte importantă). Valorile sunt memorate în format întreg, scalate prin 10 000, pentru a obține 15 cifre la partea întreagă şi 4 cifre la partea zecimală (reprezentare în virgulă fixă).

Date

Domeniu de valori: 1 ianuarie 100 — 31 decembrie 9999, 0:00:00 — 23:59:59

Memorie: 8 bytes

Declarator de tip:

Observații. Informațiile de tip dată calendaristică şi/sau timp orar sunt memorate drept numere flotante, partea întreagă reprezentând data calendaristică, partea fracționară reprezentând timpul.

La convertiri, miezul nopții este 0, miezul zilei este .5, numerele negative reprezintă date înainte de 30 decembrie 1899.

Poate fi atribuit ca valoare de tip date orice literal care reprezintă o dată calendaristică recunoscută ca atare, literalul trebuind să fie cuprins între simboluri #, de exemplu #1 Jan 99#.

Decimal

Domeniu de valori: (vezi observațiile)

Memorie: 12 bytes

Declarator de tip:

Observații. Valorile de tip Decimal sunt memorate ca întregi fără semn însoțiți de un factor de scală, între 0 şi 28, specificând numărul de zecimale. Pentru scala=0 (fără parte zecimală), cea mai

mare valoare posibilă este +/-79,228,162,514,264,337,593,543,950,335. Cu scala=28 cea mai mare valoare este +/-7.9228162514264337593543950335 iar cea mai mică valoare nenulă este +/-0.0000000000000000000000000001.

Notă: Deocamdată, tipul Decimal poate fi utilizat doar ca subtip în Variant, adică nu se pot declara variabile ca fiind de tip Decimal. Acestea pot fi create ca Variant cu subtipul Decimal prin funcția Cdec (funcția forțează o expresie să fie de un tip specificat, din aceeaşi categorie de funcții fiind şi CBool, CByte etc.).

Double

Domeniu de valori: numere negative de la -1.79769313486232E308 până la -4.94065645841247E-324; numere pozitive de la 4.94065645841247E-324 până la 1.79769313486232E308 (numere flotante în dublă precizie).

Memorie: 8 bytes

Declarator de tip: #

Observații.

Integer

Domeniu de valori: -32 768 — 32 767.

Memorie: 2 bytes

Declarator de tip: %

Observații.

Long

Domeniu de valori: -2 147 483 648 — 2 147 483 647.

Memorie: 4 bytes

Declarator de tip: &

Observații.

Object

Domeniu de valori: (vezi observațiile)

Memorie: 4 bytes

Declarator de tip:

Observații. Adrese pe 32 de biți care se referă la obiecte. Prin instrucțiunea Set se atribuie unei variabile declarate de tip Object referința la obiectul dorit.

Notă. Prin declararea unei variabile de tip Object, referirea la un obiect prin Set produce o ataşare târzie (la timpul execuției – run-time binding). Pentru o ataşare timpurie (la timpul compilării – compile-time binding) se va utiliza o variabilă declarată cu numele clasei respective.

Single

Domeniu de valori: numere negative de la -3.402823E38 până la -1.401298E-45; numere pozitive de la 1.401298E-45 până la 3.402823E38.

Memorie: 4 bytes

Declarator de tip: !

Observații.

String

Domeniu de valori: şir de lungime variabilă: până la 2^31 caractere; şir de lungime fixă: până la 2^16 caractere.

Memorie: 2 bytes

Declarator de tip: $

Observații. Un şir de lungime fixă declarat Public nu poate fi utilizat într-un modul de clasă.

Variant (default)

Domeniu de valori: aceleaşi cu domeniile specificate la tipurile precedente şi care pot fi subtipuri ale tipului Variant, cu mențiunea că toate subtipurile numerice au domeniul de la Double.

Memorie: în funcție de subtipul valorii: valorile numerice ocupă 16 bytes, valorile de tip String necesită 22 bytes plus câte un byte pentru fiecare caracter.

Declarator de tip:

Observații. Este tipul specificat implicit (în lipsa unei declarații explicite) pentru o constantă, variabilă, sau argument (caz care, deşi nerecomandat, poate elimina erorile provocate de diferențele de tip ale argumentelor la apelul procedurilor).

Cu excepția datelor de tip String cu lungime fixă şi a datelor cu tipuri definite de utilizator, tipul Variant poate conține orice alt tip de dată. În plus poate să conţină valorile speciale Empty, Error, Nothing şi Null. Tipul considerat pentru o dată conținută într-un Variant poate fi determinat cu funcția VarType sau TypeName.

Valorile unei variabile Variant pot să-şi convertească valorile automat. În general, datele numerice sunt memorate în tipul de origine, dar este posibil ca ele să fie promovate la tipul superior dacă rezultatul unei operații necesită acest fapt. De exemplu o valoare declarată inițial drept Integer şi atribuită unui Variant va fi memorată ca un întreg până când, ridicând-o de exemplu la o putere, valoarea ei excede domeniul tipului Integer. În acest caz are loc promovarea (ca mod de reprezentare) la tipul superior adecvat (Long sau Double). Dacă depăşirea domeniului are loc pentru subtipurile Currency, Decimal sau Double, atunci se va semnala eroare.

Utilizarea tipului Variant permite o tratare mai diferențiată a şirurilor de cifre: în operații numerice vor fi considerate numere iar în operații cu şiruri vor fi considerate şiruri.

Accesul la valorile Variant este mai lent decât accesul la valorile definite prin tipuri explicite.

Valorile speciale au semnificația:

Empty este valoarea unui Variant care nu a fost inițializat. În calcule numerice este considerat 0 iar în operații cu şiruri este şirul de lungime zero.

Null este valoarea unui Variant care, în mod programatic, nu conține date.

Error este valoarea utilizată pentru a arăta îndeplinirea unei condiții de eroare (prin convertirea unui real cu funcția CVErr). Procesarea se va efectua de către utilizator, tratarea automată a erorilor nu este activată la setarea acestor valori.

Nothing este utilizată pentru disocierea unei variabile de tip Object de un obiect efectiv.

Tipuri definite de utilizator

Un tip de dată definit de utilizator reprezintă echivalentul unei înregistrări dintr-un fişier (bază de date), adică o grupare de entități de tipuri diferite. Definirea are loc la nivel de modul, prin instrucțiuni Type. Pentru clauzele care apar se va vedea discuția de la domeniul variabilelor.

[Private | Public] Type *varname*

elementname **[(**[subscripts]**)] As** type

[elementname **[(**[subscripts]**)] As** type**]**

. . .

End Type

unde *varname* este numele dat tipului definit, iar prin elementname se definesc componentele tipului. Se pot utiliza şi componente de tipuri utilizator deja definite. Componentele pot fi şi tablouri, caz în care apar definițiile specifice (vezi declararea variabilelor).

Declararea constantelor, variabilelor şi tablourilor

Nume

La denumirea procedurilor, constantelor, variabilelor şi argumentelor într-un modul Visual Basic

se cere respectarea următoarelor reguli:

primul caracter trebuie să fie o literă;

nu se utilizează spațiu, punct (.), semnul exclamării(!), sau caracterele @, &, $, #

lungimea denumirii nu poate depăşi 255 de caractere;

la acelaşi nivel de existență nu pot să existe denumiri identice. Pot să existe totuşi, în acelaşi modul, o variabilă privată şi o variabilă la nivel de procedură care să poarte acelaşi nume.

În general, nu se recomandă definirea unor denumiri identice cu nume de funcții, instrucțiuni sau metode existente în Visual Basic. Dacă s-a ajuns totuşi la această situație, atunci utilizarea funcției intrinseci limbajului, a instrucțiunii sau metodei care intră în conflict cu un nume asignat necesită calificarea ei în raport de biblioteca asociată. De exemplu, VBA.Left este apelul la funcția Left atunci când este definită de utilizator şi o variabilă Left. Notă. Visual Basic nu este *case-sensitive*, deci denumirea unei entități nu are ca atribut distinctiv capitalizarea literelor, dar mediul de programare VBA păstrează capitalizarea din instrucțiunea unde este definit un nume.

Declararea constantelor

Definirea unei constante se realizează prin instrucțiunea Const, în care se poate specifica tipul, domeniul şi valoarea constantei. Valoarea unei constante nu se poate schimba programatic.

[Public | Private] Const constname **[As** type**] =** expression

Public — cuvânt cheie, opțional, utilizat la nivel de modul pentru a declara constante recunoscute în toate procedurile din toate modulele. Nu este permis în proceduri.

Private — cuvânt cheie, opțional, utilizat la nivel de modul pentru a declara constante recunoscute în toate procedurile din modulul în care apare declarația. Nu este permis în proceduri.

constname — numele constantei (obligatoriu).

type — tipul constantei: Byte, Boolean, Integer, Long, Currency, Single, Double, Decimal (încă nu este suportat), Date, String, sau Variant. Fiecare constantă presupune o clauză As type proprie; în lipsa clauzei se va ataşa automat tipul cel mai apropiat expresiei.

expression — combinație de identificatori, constante, operatori (cu excepția Is) care produce un sir, număr sau obiect. Nu se pot utiliza variabile, funcții utilizator sau funcții VBA predefinite.

În mod implicit, constantele sunt private. La nivel de procedură, sau de modul clasă, domeniul lor nu poate fi modificat prin utilizarea clauzei Public. La nivel de modul standard vizibilitatea poate fi modificată prin Public.

Constantele declarate în proceduri Sub, Function sau Property sunt locale procedurii, constantele declarate în afara unei proceduri este definită în modulul respectiv.

Mai multe declarații de constante pot fi scrise pe o aceeaşi linie, separate prin virgule la nivel de atribuiri de expresii. În acest caz, cuvintele Public sau Private care apar se aplică întregii linii.

Exemple

```
Const NrLinii = 15

Public Const MesajInitial = "Tastati numarul de linii"

Private Const NrLinii as Integer = 15

Public Const NrLinii = 15, Pondere as Single = 1.21
```

Este de remarcat că, în ultima linie, doar Pondere este de tip Single, în timp ce NrLinii este de tip Integer (în lipsa clauzei As type se atribuie tipul expresiei).

Declararea variabilelor

Variabilele, simple sau tablou, se definesc prin instrucțiunile Dim, Private, Public, ReDim sau Static. Numele unei variabile trebuie să respecte regulile generale de formare a identificatorilor, tipul variabilei poate fi definit explicit (prin clauza As type) sau implicit (ca Variant).

În cazul în care modulul conține instrucțiunea Option Explicit cu sintaxa

Option Explicit

şi care trebuie să apară înaintea oricărei proceduri din modul, toate variabilele trebuie să fie declarate prin instrucțiunile menționate. Lipsa instrucțiunii Option Explicit permite ca variabilele să fie definite acolo unde este nevoie de ele prin simpla menționare a unui nou identificator, tipul lor fiind stabilit implicit. Această ultimă posibilitate poate produce erori greu detectabile.

Sintaxa instrucțiunilor de declarare a variabilelor este următoarea şi se observă asemănarea clauzelor.

Dim [WithEvents] varname**[**([subscripts])**]** **[As [New]** type**]**

Private [WithEvents] varname**[**([subscripts])**]** **[As [New]** type**]**

Public [WithEvents] varname**[**([subscripts])**]** **[As [New]** type**]**

Static varname**[**([subscripts])**]** **[As [New]** type**]**

ReDim [Preserve] varname(subscripts) **[As** type**]**

varname — numele variabilei (obligatoriu).

subscripts — dimensiunile tabloului de date (dacă se declară o variabilă tablou). Pot exista până la 60 de indici, separați prin virgule, declararea dimensiunilor pentru un indice fiind de forma

> [lower **To**] upper
>
> Limita inferioară este, implicit, 0, dar poate fi controlată prin instrucțiunea Option Base.
>
> Dacă nu se indică limitele indicilor (dar parantezele sunt prezente), se defineşte o variabilă tablou dinamică (nu şi prin **Static**) ale cărei dimensiuni pot fi precizate/redefinite prin instrucțiunea **ReDim**.

New — permite crearea implicită a unui obiect (atunci când se declară o variabilă de tip obiect). O nouă instanță a obiectului este creată la prima referință a variabilei definite. Clauza nu poate să apară la declararea variabilelor de tipuri intrinseci şi nici la declararea instanțelor obiectelor dependente.

type — tipul variabilei definite: Byte, Boolean, Integer, Long, Currency, Single, Double, Decimal (nesuportat încă), Date, String (pentru şiruri cu lungime variabilă), String * length (pentru şiruri cu lungime fixă), Object, Variant, tip utilizator sau tip de obiect.

Dacă se definesc mai multe variabile într-o instrucțiune, definițiile se separă prin virgulă iar clauza de tip nu este extinsă şi la variabilele definite ulterior.

Deşi toate instrucțiunile permit declararea unor variabile (simple sau tablou), fiecare instrucțiune are un efect distinct în ceea ce priveşte vizibilitatea variabilelor şi persistența valorilor.

Dim defineşte variabile atât la nivel de modul cât şi la nivel de procedură. Variabilele definite la nivel de modul sunt accesibile în procedurile acelui modul, iar variabilele de la nivel de procedură sunt vizibile doar în procedura respectivă.

Private este utilizată la nivel de modul pentru a declara variabile accesibile doar în procedurile acelui modul.

Public este utilizată pentru a declara variabile accesibile în toate procedurile din toate modulele şi din toate aplicațiile. Prin includerea instrucțiunii Option Private Module este posibil ca variabilele publice să fie vizibile doar în proiectul în care sunt definite.

Static este utilizată la nivel de procedură nestatică pentru a declara variabile care îşi păstrează valoarea de la o execuție a procedurii la alta, atât timp cât modulul în care apare procedura nu este resetat sau repornit. Variabilele definite prin Static sunt vizibile doar în procedura respectivă. Este de remarcat că se poate defini o întreagă procedură utilizând clauza Static (vezi definirea procedurilor), caz în care toate variabilele sunt statice.

ReDim este utilizată la nivel de procedură pentru realocarea memoriei variabilelor tablou dinamice. Utilizarea clauzei Preserve permite doar modificarea ultimei dimensiuni şi păstrează valorile deja existente. (Pentru detalii vezi şi VBA Help).

Exemple

```
Dim x As Double, ColtStanga As Integer

Private I, J As Long

Static Venit As Currency, NumPren As String

Dim Retineri(5) As Currency

Public indicatori(10) As Byte

Dim matrice(1 To 3, 100 To 200) As String

Public fntScris As Font

Dim appWD As Word.Application
```

Proceduri

Printr-o procedură se înțelege, similar altor limbaje de programare, o mulțime de instrucțiuni care este identificată printr-un nume şi care se execută unitar printr-un singur apel. Ar trebui, pentru claritatea programului, ca o procedură să efectueze o prelucrare unitară identificabilă în logica programului.

Există trei tipuri principale de proceduri: Sub, Function şi Property. Ultimul tip este caracteristic definirii unui obiect şi va fi prezentat ulterior. O procedură de tip Sub poate primi şi transmite informații prin intermediul unor variabile publice sau/şi a unor parametri. Numele procedurii nu are ataşată nici o valoare. O procedură de tip Function se deosebeşte prin aceea că numele procedurii are ataşată o valoare (valoarea funcției) şi poate fi utilizat ca orice altă variabilă din proiect.

Observație. Orice instrucțiune executabilă trebuie să aparțină unei proceduri. Declarațiile pot să apară şi în afara procedurilor, la nivel de modul.

Proceduri Sub

Organizarea generală a unei proceduri de tip Sub este

[Private | Public] [Static] Sub name **(**[arglist]**)**

    [instrucțiuni**]**

    [Exit Sub]

    [instrucțiuni**]**

End Sub

Public, Private, Static — determină vizibilitatea procedurii. Public = vizibilă pentru toate procedurile şi toate modulele (în funcție de Option Private se defineşte vizibilitatea pentru alte proiecte). Private = vizibilă doar pentru procedurile din modulul unde procedura este declarată. Static = arată că toate variabilele locale îşi păstrează valorile între apeluri.

name — numele procedurii.

arglist — lista de argumente, separate prin virgule.

Prin instrucțiunea Exit Sub se poate ieşi din procedură şi altminteri decât prin linia finală.

Argumentele se definesc după sintaxa:

[Optional] [ByVal | ByRef] [ParamArray] varname**[(** **)] [As** type**] [=** defaultvalue**]**

Optional — arată că parametrul nu este obligatoriu. Parametrii opționali trebuie să fie grupați la sfârşitul listei (apariția clauzei Optional cere ca toți parametrii care urmează să aibă aceeaşi clauză).

ByVal — arată că apelul paramatrului se face prin valoare (orice modificare a valorii transmise nu este regăsită după părăsirea procedurii, calculele efectuându-se pe o copie a parametrului).

ByRef — arată că apelul paramatrului se face prin referință (orice modificare a valorii transmise este regăsită după părăsirea procedurii). Acesta este modul implicit de transmitere a parametrilor.

ParamArray — folosit doar ca ultim argument în listă, denotă un tablou Optional de elemente de tip Variant. Clauza ParamArray permite definirea unui număr arbitrar de parametri. ParamArray nu poate fi utilizat împreună cu ByVal, ByRef, sau Optional.

varname — numele argumentului. Dacă este tablou se vor indica parantezele.

type — tipul parametrului transmis: Byte, Boolean, Integer, Long, Currency, Single, Double, Decimal, Date, String (doar lungime variabilă), Object, Variant. Pentru parametrii obligatorii (fără Optional) poate fi şi un tip definit sau de obiect.

defaultvalue — defineşte valoarea implicită pentru argumentele opționale. Poate fi orice expresie, dar pentru tipul Object se admite doar Nothing.

Apelul unei proceduri Sub

Pentru a executa o procedură de tip Sub din altă procedură (vezi şi discuția privind vizibilitatea) se menționează, pe o linie separată, numele procedurii urmat sau nu de parametri. Dacă este necesar, datorită apelării unui alt proiect sau modul, atunci apelul este după modelul:

Nume_proiect**.**Nume_modul**.**Nume_procedură listă de argumente

unde lista de argumente poate sau nu să fie inclusă între paranteze. Argumentele efective sunt separate în listă prin virgule şi trebuie să respecte ordinea (şi tipul) argumentelor din definiția procedurii. În cazul procedurilor cu multe argumente, dintre care multe opționale, transferul poate provoca erori de scriere a codului (un argument opțional necesită totuşi virgula sa, de unde o numărare atentă a virgulelor etc.). Pentru asemenea situații (în special) se permite şi transferul valorilor prin intermediul tehnicii de argumente denumite. Aceasta se realizează alcătuind lista de argumente, la apelul procedurii, din intrări de forma

nume_argument**:=**valoare_argument

separate prin virgule şi la care nu mai contează ordinea inițială a argumentelor. Se vor specifica doar parametrii care se transmit efectiv (adică valorile opționale dorite şi toate valorile neopționale).

Pentru apelul unei proceduri se va studia şi instrucțiunea Call.

Proceduri Function

O procedură de tip Function este similară, ca definiție, unei proceduri Sub, dar are particularitatea că returnează o valoare prin numele său (care se comportă deci ca o variabilă).

[Public | Private] [Static] Function name [(arglist)] **[As** type**]**

[statements]

[name = expression]

[Exit Function]

[statements]

[name = expression]

End Function

Este de remarcat că se poate ataşa un tip numelui funcției (adică valorii funcției) şi se va remarca existența instrucțiunilor prin care se atribuie funcției valorile calculate.

Valoarea returnată de o funcție poate fi utilizată într-o altă expresie prin includerea numelui funcției urmat, între paranteze, de valorile efective ale parametrilor.

Dacă apelul se face prin intermediul instrucțiunii Call, valoarea funcției nu poate fi utilizată. În asemenea situații se activează de fapt doar prelucrările colaterale (care, pentru claritatea codului, nici nu sunt recomandate).

Exemple de proceduri

```
Public Function AriaCilindru (raza, inaltime) As Double
        Const Pi = 3.14159
        cilBaza = Pi*raza^2
        cilLaterala = 2*Pi*raza*inaltime
        AriaCilindru = 2*cilBaza + cilLaterala
End Function


Sub AriaCilindru (ByVal raza As Single, ByVal inaltime As Single, ByRef cilAria As Double)
        Const Pi As Single = 3.14159
        Dim cilBaza As Single, cilLaterala As Single
        cilBaza = Pi*raza^2
        cilLaterala = 2*Pi*raza*inaltime
        cilAria = 2*cilBaza + cilLaterala
End
```

Apelul funcției poate fi într-o instrucțiune de genul

```
CostTotalPiesa = AriaCilindru (r1, h1) * CostUnitar
```

în timp ce apelul subrutinei poate fi

```
AriaCilindru inaltime:=h1, raza:=r1, cilAria:=AriePiesa
```

Organizarea generală a unui proiect VBA

Obiectele şi prelucrările necesare realizării unei aplicații VBA (presupunând că se doreşte atingerea unui ansamblu coerent de scopuri) sunt gestionate sub forma unui proiect, care are un nume implicit sau dat de utilizator. La un anumit moment pot fi deschise mai multe proiecte, identificabile prin denumirile lor.

Deoarece prelucrările proiectate în VBA sunt ataşate documentelor (acțiunilor) unor aplicații particulare (Word, Excel etc.), proiectele sunt salvate o dată cu documentele pe care le însoțesc. Acest fapt nu reduce aria de probleme abordabile întrucât prelucrările propriu-zise nu sunt limitate la documentul însoțit (se poate deschide astfel un document Word alb şi să se efectueze orice prelucrare dorită, fără a avea obligația de a scrie ceva în documentul deschis).

Într-un proiect VBA sunt identificabile următoarele componente:

Module standard (denumite inițial module de cod). Conțin declarații şi proceduri generale. Există de asemenea şi module care conțin tratarea evenimentelor specifice documentului de care este ataşat proiectul.

Module de clasă. Conțin definirea obiectelor create de utilizator.

Forme. Conțin definițiile dialogurilor din interfața proiectată de utilizator ca şi codul program necesar controlării dialogurilor.

Referințe. Într-un proiect este menținută lista altor proiecte, care sunt referite în proiectul curent.

Un modul de cod poate începe cu o secțiune de declarații. Prin declarații înțelegem instrucțiuni neexecutabile prin care se definesc constante, variabile şi proceduri externe. Utilizând Public, Static, Private se precizează şi domeniul de vizibilitate a entităților definite.

Gestionarea (crearea, editarea, ştergerea etc.) obiectelor dintr-un proiect se face prin comenzi ale mediului VBA, care este prezentat într-o secțiune separată.

Domeniul unei variabile, constante sau proceduri

Domeniul unei entități reprezintă mulțimea instrucțiunilor unde poate fi referită acea entitate. Se poate vorbi astfel de vizibilitatea unei entități. Domeniul este dependent de locul definirii entității, de clauzele care apar la definire şi de parametrii globali ai proiectului.

> **Notă.** Este de remarcat că utilizarea unei denumiri în afara domeniului inițial prefigurat produce, în lipsa instrucțiunii Option Explicit, crearea unei noi entități, fără nici o legătură cu cea precedentă, sursă de erori greu detectabile. Acesta este motivul pentru care se recomandă declararea explicită a tuturor variabilelor.

Există trei tipuri de domenii:

la nivel de procedură;

la nivel de modul, privat;

la nivel de modul, public.

Nivelul procedură

O variabilă sau constantă definită într-o procedură este vizibilă doar în procedură respectivă. Dacă o asemenea entitate trebuie referită şi în alte proceduri, atunci declararea ei se va efectua la nivel de modul, sau se va transmite procedurii prin intermediul argumentelor.

Nivel de modul, privat

Variabilele şi constantele definite la nivel de modul (în secțiunea Declarations) sunt Private în mod implicit, adică sunt vizibile doar în modulul respectiv. Utilizarea clauzei Private nu este deci necesară, dar este recomandată.

Notă. Dacă se utilizează instrucțiunea Option Private Module (în secțiunea Declarations a modulului) atunci variabilele şi procedurile publice vor fi vizibile doar în proiectul curent. În lipsa acestei declarații, procedurile publice (din toate modulele standard sau clasă) sunt vizibile în toate proiectele care se referă la proiectul curent. Procedurile, variabilele şi constantele publice din alte module (cum ar fi modulele ataşate formelor) sunt Private pentru proiectul de definiție, deci ele nu sunt accesibile proiectelor care se referă la proiectul unde sunt declarate.

Nivel de modul, public

Variabilele declarate la nivel de modul drept Public sunt vizibile în toate procedurile din proiect. Procedurile sunt publice în mod implicit, cu excepție procedurilor de tratare a evenimentelor, care sunt Private în mod implicit. A se vedea şi nota anterioară.

Viata unei variabile

Prin viața unei variabile se înțelege timpul cât variabila are o valoare. Este evident că valoarea unei variabile se poate modifica pe durata vieții sale, dar definitoriu este faptul că variabila are o anumită valoare pe întreaga durată a vieții sale. La părăsirea domeniului, variabila "moare" şi nu mai are ataşată o valoare.

La începutul execuției unei proceduri, toate variabilele sunt inițializate:

Variabilă numerică	0 (zero)
Şir de lungime variabilă	"" (şir de lungime zero)
Şir de lungime fixă	Completat cu caracterul Chr(0) (având codul ASCII 0)
Variabilă Variant	Empty
Variabile de tip utilizator	fiecare element este inițializat separat, potrivit tipului primar
Variabilă Object	Nothing (până la asignarea unei referințe prin Set)

Variabilele care nu sunt modificate îşi păstrează valoarea inițială.

Variabilele declarate prin Dim la nivel de procedură au valoare până la terminarea execuției procedurii (chiar dacă se trece prin apel în alte proceduri).

Variabilele declarate prin Static, la nivel de procedură, au aceeaşi viață ca şi variabilele declarate la nivel de modul şi îşi păstrează valoarea până la terminarea execuției codului (inclusiv de la un apel la altul). Includerea clauzei Static în instrucțiunea Sub sau Function are ca efect declararea tuturor variabilelor definite în procedura respectivă drept variabile statice (deci care îşi păstrează valorile între apeluri).

Variabilele declarate la nivel de modul standard îşi păstrează valoarea pe tot timpul execuției. Variabilele declarate la nivel de modul clasă îşi păstrează valoarea atât timp cât există o instanță a clasei. Diferența față de variabilele Static este aceea că memoria este utilizată permanent (nu se eliberează la părăsirea domeniului).

Variabile Object

Declararea unei variabile de tip obiect se poate efectua prin declararea tipului generic Object

```
Dim myDoc As Object
```

sau specificând exact numele de clasă dintr-o bibliotecă de obiecte referită

```
Dim myDoc As Word.Document
```

În primul mod de definire (ca Object) nu se poate efectua la momentul compilării existența obiectului, nu se poate verifica utilizarea corectă a proprietăților şi metodelor obiectului şi nu se poate lega această informație de variabila obiect definită. Ataşarea unui obiect este, în acest caz, o legare târzie (*late binding*) la momentul execuției şi se efectuează prin instrucțiunea Set.

Specificarea unei clase la definirea variabilei obiect produce o legare timpurie (*early binding*) care este mai rapidă, se face la momentul compilării şi poate înlătura mai rapid erori posibile în utilizarea metodelor şi proprietăților obiectului.

Instrucțiunea Set are sintaxa:

Set objectvar = **{[New]** objectexpression | **Nothing}**

unde

objectvar este numele variabilei (sau proprietății)

New permite crearea unei noi instanțe a clasei

Objectexpression este o expresie constând în numele unui obiect, altă variabilă declarată de acelaşi tip obiect, sau funcție ori metodă care returnează un obiect de acelaşi tip obiect

Nothing permite deconectarea asocierii cu un obiect specific, eliberând resursele sistem şi de memorie utilizate.

În general, atunci când se utilizează Set pentru a asigna o referință de obiect la o variabilă, nu se creează o copie a obiectului pentru acea variabilă. Este creată doar o referință la obiect. Astfel, mai multe variabile de tip obiect pot să se refere la acelaşi obiect: orice schimbare a obiectului se va reflecta în toate variabilele care referă obiectul. Utilizând clauza New se va crea efectiv o copie (instanță) a obiectului.

Exemple

Prin următoarele două instrucțiuni se defineşte variabila objWord care este legată târziu de o aplicație Word:

```
Dim objWord As Object
Set obhWord = CreateObject("Word.Application")
```

Legarea timpurie se poate efectua prin

```
Dim objWord As Word.Application
```

Este de remarcat că instrucțiunea Set apelează o funcție care creează şi returnează o referință la un obiect ActiveX.

Constante predefinite (built–in)

Bibliotecile de obiecte din fiecare aplicație Office furnizează o mulțime de constante predefinite, care pot fi utilizate pentru a stabili proprietăti sau pentru a transmite argumente către proprietăți sau metode. Constantele sunt, de regulă, grupate în tipuri enumerate care reprezintă valorile posibile pentru o proprietate specifică. Deşi este posibilă să se utilizeze valoarea numerică a constantei este recomandat să se utilizeze constanta numită întrucât dezvoltări ulterioare ale mediului Microsoft Office (ca şi ale aplicațiilor din Visual Studio) tind să păstreze compatibilitatea între denumirile constantelor şi nu între valorile efective.

De exemplu se preferă

```
Application.DisplayAlerts = wdAlertAll
```

în loc de

```
Application.DisplayAlerts = -1
```

pentru a fixa ca Word să afişeze toate mesajele de alertă la execuția unei proceduri. Codul scris astfel este şi mai explicit.

Instrucțiunile VBA

Există trei categorii de instrucțiuni Visual Basic:

instrucțiuni de declarare (prezentate la declararea variabilelor) prin care se denumesc şi se declară tipul pentru variabile, constante şi proceduri;

instrucțiuni de atribuire (prezentate în continuare) prin care se atribuie valori variabilelor sau constantelor;

instrucțiuni executabile (prezentate în continuare) care inițiază acțiuni: execută metode sau proceduri, controlează fluxul execuției codului.

În mediul de dezvoltare VBA, sintaxa instrucțiunilor este verificată automat după ce se trece la instrucțiunea următoare (prin Enter).

Continuarea instrucțiunilor

O instrucțiune poate să fie scrisă pe mai multe linii prin utilizarea caracterului de continuare a liniei "_" precedat de un spațiu. De exemplu, crearea prin program a unui tabel într-un document Word:

```
ActiveDocument.Tables.Add Range:=Selection.Range, _
NumRows:=3, _
NumColumns:= 3
```

unde, pe lângă continuarea liniilor se va remarca utilizarea argumentelor numite la apelul metodei de adăugare a unui nou tabel la colecția de tabele a documentului.

Două instrucțiuni pot fi scrise pe o aceeaşi linie dacă sunt separate cu caracterul ":".

Etichetarea liniilor

O linie poate fi identificată:

printr-o etichetă: orice nume, care respectă regulile generale, care începe în prima coloană a liniei şi se termină cu caracterul ":"

printr-un număr: orice combinație de cifre, care începe în prima coloană a liniei şi este unic în modulul respectiv.

Identificatorii de linii pot fi utilizați în instrucțiuni de control, desi codul astfel construit nu respectă regulile programării structurate..

Comentarii

Textele explicative (necesare documentării codului) pot fi introduse pe linii separate sau în continuarea liniei de cod.

O linie de comentariu începe cu un apostrof (') sau cu cuvântul **Rem** urmat de un spațiu.

Comentariul de pe aceeaşi linie cu o instrucțiune se introduce printr-un apostrof urmat de comentariu.

Operatori

În formarea expresiilor de diverse tipuri, operatorii sunt cei utilizați aproape general în limbajele de programare de nivel înalt. Pentru fixarea termenilor şi notațiilor sunt totuşi prezentați, pe categorii, însoțiți, acolo unde este cazul de scurte explicații.

Operatori aritmetici

Operator	Semnificație	Observații
^	Ridicarea la putere	rezultatul este Double sau Variant(Double) cu excepția: dacă un operand este Null, rezultatul este tot Null
*	Înmulțirea	rezultatul este dat de cel "mai precis" factor, ordinea crescătoare a "preciziei" fiind, pentru înmulțire, Byte, Integer, Long, Single, Currency, Double şi Decimal. Dacă o expresie este Null, rezultatul este Null. O expresie Empty este considerată ca 0. Pentru excepții se va studia Help – *(operator).
/	Împărțirea	rezultatul este, în general, Double sau Variant(Double). Dacă o expresie este Null, rezultatul este Null. O expresie Empty este considerată ca 0. Pentru excepții se va studia Help – /(operator).
\	Împărțirea întreagă	înainte de împărțire, operanzii sunt rotunjiți la Byte, Integer sau Long. Rezultatul este Byte, Variant(Byte), Integer, Variant (Integer), Long, sau Variant(Long). Dacă o expresie este Null, rezultatul este Null. O expresie Empty este considerată ca 0.
Mod	Restul împărțirii	operanzii sunt rotunjiți la întregi şi se obține restul împărțirii. Rezultatul este Byte, Variant(Byte), Integer, Variant (Integer), Long, sau Variant(Long). Dacă o expresie este Null, rezultatul este Null. O expresie Empty este considerată ca 0.
+	Adunarea numerică sau concatenarea şirurilor	în general, operanzi numerici produc adunarea, iar operanzi şiruri produc concatenarea. În cazul numeric, rezultatul este de tipul cel "mai precis" al operanzilor, ordinea de "precizie" fiind pentru adunare şi scădere: Byte, Integer, Long, Single, Double, Currency şi Decimal. Deoarece operanzii pot fi orice expresie, pentru o informare completă (de exemplu operanzi Variant) se va studia Help

		– +(operator).
-	Scăderea sau inversarea semnului	operanzii pot fi doar numerici. Rezultatul este de tipul cel "mai precis" al operanzilor, ordinea de "precizie" fiind pentru adunare şi scădere: Byte, Integer, Long, Single, Double, Currency şi Decimal. Dacă o expresie este Null, rezultatul este Null. O expresie Empty este considerată ca 0. Pentru excepţii se va studia Help – -(operator).

Operatori de comparare

Relaţiile care există între diferite tipuri de entităţi se pot evidenţia prin comparaţii având una dintre formele următoare:

result = expression1 **comparisonoperator** expression2

result = object1 **Is** object2

result = string **Like** pattern

unde

result este o variabilă numerică

expression este o expresie oarecare

comparisonoperator este un operator relaţional

object este un nume de obiect

string este o expresie şir oarecare

pattern este o expresie String sau un domeniu de caractere.

Operatorii de comparare sunt cei uzuali: < (mai mic), <= (mai mic sau egal), > (mai mare), >= (mai mare sau egal), = (egal), <> (diferit, neegal).

Rezultatul este True (dacă este adevărată relaţia), False (dacă relaţia este neadevărată), Null (dacă cel puţin un operand este Null).

Operatorul **Is** produce True dacă variabilele se referă la acelaşi obiect şi False în caz contrar.

Operatorul **Like** compară două şiruri cu observaţia că al doilea tremen este un şablon. Prin urmare rezultatul este True dacă primul şir operand este format după şablon, False în caz contrar. Atunci când un operand este Null, rezultatul este tot Null.

Comportarea operatorului Like depinde de instrucţiunea Option Compare, care poate fi:

Option Compare Binary, ordinea este cea a reprezentării interne binare, determinată în Windows de codul de pagină.

Option Compare Text, compararea este insenzitivă la capitalizarea textului, ordinea este determinată de setările locale ale sistemului.

Construcţia şablonului poate cuprinde caractere wildcard, liste de caractere, domenii de caractere:

? un caracter oarecare

* oricâte caractere (chiar nici unul)

o cifră oarecare (0–9).

[charlist] oricare dintre caracterele enumerate în listă, un domeniu de litere poate fi dat prin utilizarea cratimei.

[!charlist] orice caracter care nu este în listă

Observaţie. Pentru a utiliza în şablon caracterele speciale cu valoare de wildcard se vor utiliza construcţii de tip listă: [[], [?] etc. Paranteza dreapta va fi indicată singură:].

Pentru alte observaţii utile se va studia Help – Like operator.

Operatori de concatenare

Pentru combinarea şirurilor de caractere se pot utiliza operatorii & şi +.

În sintaxa

expression1 & expression2

unde operanzii sunt expresii oarecare, rezultatul este

de tip String, dacă ambii operanzi sunt String

de tip Variant(String) în celelalte cazuri

Null, dacă ambii operanzi sunt Null.

Înainte de concatenare, operanzii care nu sunt şiruri se convertesc la Variant(String). Expresiile Null sau Empty sunt tratate ca şiruri de lungime zero ("").

Operatori logici

Pentru operațiile logice sunt utilizați următorii operatori, uzuali în programare.

Operator	Semnificație	Observații
And	conjuncția logică	Null cu False dă False, Null cu True sau cu Null dă Null. Operatorul And realizează şi operația de conjuncție bit cu bit pentru expresii numerice.
Eqv	echivalența logică	Dacă o expresie este Null, rezultatul este Null. Eqv realizează şi compararea bit cu bit a două expresii numerice, poziționând cifrele binare ale rezultatului după regulile de calcul ale echivalenței logice: 0 Eqv 0 este 1 etc.
Imp	implicația logică	True Imp Null este Null, False Imp * este True, Null Imp True este True, Null Imp False (sau Null) este Null. Operatorul Imp realizează şi compararea bit cu bit a două expresii numerice, poziționând cifrele binare ale rezultatului după regulile de calcul ale implicației logice: 1 Imp 0 este 0, în rest rezultatul este 1.
Not	negația logică	Not Null este Null. Prin operatorul Not se poate inversa bit cu bit valorile unei variabile, poziționându-se corespunzător un rezultat numeric.
Or	disjuncția logică	Null Or True este True, Null cu False (sau Null) este Null. Operatorul Or realizează şi o comparație bit cu bit a două expresii numerice poziționând biții corespunzători ai rezultatului după regulile lui Or logic.
Xor	disjuncția exclusivă	Dacă un operand este Null, atunci rezultatul este Null. Se poate efectua operația de sau exclusiv şi bit cu bit pentru două expresii numerice [b1+b2(mod 2)].

Instrucțiuni de atribuire

Atribuirea se poate efectua prin instrucțiunea Let (pentru valori atribuite variabilelor şi proprietăților), Set (pentru atribuirea de obiecte la o variabilă de tip obiect), Lset şi Rset (pentru atribuiri speciale de şiruri sau tipuri definite de utilizator).

Instrucțiunea Let

Atribuie valoarea unei expresii la o variabilă sau proprietate.

[Let] varname **=** expression

unde *varname* este nume de variabilă sau de proprietate.

Este de remarcat forma posibilă (şi de fapt general utilizată) fără cuvântul Let.

Observații. Valoarea expresiei trebuie să fie compatibilă ca tip cu variabila (sau proprietatea): valori numerice nu pot fi atribuite variabilelor de tip String şi nici reciproc.

Variabilele Variant pot primi valori numerice sau String, reciproc nu este valabil decât dacă valoarea expresiei Variant poate fi interpretată compatibilă cu tipul variabilei: orice Variant poate fi atribuit unei variabile de tip String (cu excepția Null), doar Variant care poate fi interpretat nuric poate fi atribuit unei variabile de tip numeric.

La atribuirea valorilor numerice pot avea loc conversii la tipul numeric al variabilei.

Atribuirea valorilor de tip utilizator poate fi efectuată doar dacă ambii termeni au acelaşi tip definit. Pentru alte situații se va utiliza instrucțiunea Lset.

Nu se poate utiliza Let (cu sau fără cuvântul Let) pentru legarea de obiecte la variabile obiect. Se va utiliza în această situație instrucțiunea Set.

Instrucțiunea LSet

Copie, cu aliniere la stânga, un şir de caractere (valoarea expresiei din dreapta) într-o variabila de tip String. Deoarece copierea este binară, poate fi utilizată pentru atribuiri între tipuri utilizator diferite (rezultatul este impredictibil deoarece nu se face nici o verificare de tipuri/componente ale valorilor de tip record). Sintaxa este

LSet stringvar **=** string

LSet varname1 **=** varname2

unde

stringvar, string reprezintă variabila de tip String şi expresia de acelaşi tip implicate într-o atribuire de şiruri.

varname1, varname2 sunt denumiri de variabile, de tipuri definite de utilizator (vezi instrucțiunea Type) diferite. Zona de memorie alocată celei de a doua variabile este copiată (aliniată la stânga) în zona de memorie a primei variabile.

Caracterele care rămân neocupate se completează cu spații, iar dacă zona de unde se copie este mai mare, caracterele din dreapta se pierd (sunt trunchiate).

Instrucțiunea LSet

Copie, cu aliniere la dreapta, un şir de caractere (valoarea expresiei din dreapta) într-o variabila de tip String. Sintaxa este

RSet stringvar **=** string

Caracterele rămase neocupate în variabilă sunt completate ccu spații. Instrucțiunea RSet nu se poate utiliza (analog lui LSet) pentru tipuri definite de utilizator.

Instrucțiuni executabile

Execuția unui program are loc, în lipsa oricărui control, instrucțiune cu instrucțiune, de la stânga la dreapta şi de sus în jos. Acest sens poate fi modificat, într-o oarecare măsură, prin ordinea de precedență a operațiilor în evaluarea expresiilor. Este evident că o asemenea structură simplă nu poate cuprinde toate aspectele programării şi din acest motiv necesitatea structurilor de control a fluxului execuției. Unele instrucțiuni au fost păstrate doar din motive de compatibilitate cu versiunile inițiale ale limbajului, în locul lor fiind preferate structuri mai evoluate sau similare altor limbaje de programare.

Instrucțiuni de transfer (GoSub…Return, GoTo, OnError, On…GoSub, On…GoTo)

Această categorie cuprinde instrucțiunile prin care controlul execuției este transferat la o altă instrucțiune din procedură. În general, utilizarea acestor comenzi nu produce programe foarte structurate (în sensul programării structurate) şi prin urmare, pentru o mai mare claritate a codului, pot fi înlocuite cu alte structuri de programare.

GoSub…Return

În cadrul unei proceduri un grup de instrucțiuni poate fi organizat ca o subrutină (similar unei proceduri on-line, nenumite) identificată prin linia de început. Transferul controlului la acest grup de instrucțiuni şi revenirea la locul apelului se poate efectua prin GoSub…Return cu sintaxa

GoSub line

...

line

...

Return

unde *line* este o etichetă de linie sau un număr de linie din aceeaşi procedură.

Pot exista mai multe instrucțiuni Return, prima executată produce saltul la instrucțiunea care urmează celei mai recente instrucțiuni GoSub executate.

GoTo

Realizează tranferul controlului execuției la o linie din aceeaşi procedură.

GoTo line

unde *line* este o etichetă de linie sau un număr de linie din aceeaşi procedură.

On Error

Permite controlul erorilor prin transferul controlului la rutine de tratare.

Observație. Este prezentată în secțiunea dedicată controlului erorilor.

On…GoSub, On…GoTo

Permit o ramificare multiplă, după valoarea unei expresii. Se recomandă, pentru claritatea codului, utilizarea structurii Select Case în locul acestor structuri.

On expression **GoSub** destinationlist

On expression **GoTo** destinationlist

unde

expression este o expresie numerică având valoare întreagă (după o eventuală rotunjire) între 0 şi 255 inclusiv.

destinationlist este o listă de etichete de linii sau numere de linii, separate prin virgule (elementele pot fi de ambele categorii), din aceeaşi procedură cu instrucțiunea.

Dacă valoarea expresiei este negativă sau mai mare decât 255 se produce o eroare.

Dacă valoarea expresiei, fie ea *k*, este în domeniul rangurilor listei, atunci se transferă controlul la linia identificată de al *k*-lea element al listei.

Dacă valoarea expresiei este 0 sau mai mare decât numărul de elemente din listă, transferul se efectuează la linia care urmează instrucțiunea On...GoSub sau On...GoTo.

Instrucțiuni de terminare sau oprire a programului (DoEvents, End, Exit, Stop)

Terminarea execuției programului sau oprirea temporară (pauza) se pot realiza prin instrucțiunile enumerate aici.

DoEvents

Deşi nu este o instrucțiune VBA ci este o funcție, includerea ei este naturală prin aceea că permite cedarea controlului către sistemul de operare, care poate astfel să funcționeze în regim de multitasking. Acțiunea poate fi realizată şi prin alte tehnici (de exemplu utilizarea unui Timer etc.). Sintaxa este

DoEvents( )

Funcția returnează, în general, valoarea 0.

Controlul este redat programului după ce sistemul de operare a terminat procesarea evenimentelor din coada de evenimente, ca şi procesarea tuturor caracterelor din coada SendKeys.

Observație. Pentru alte observații se va studia documentația comenzii DoEvents.

End

Termină execuția unei proceduri (sub forma prezentată aici) sau indică sfârşitul codului unei structuri de tip bloc (cum ar fi End Function, End If etc., prezentate la structurile respective).

Sintaxa, în ipostaza opririi execuției, este:

End

Prin această instrucțiune, care poate fi plasată oriunde în program, execuția este terminată imediat, fără a se mai executa eventualele instrucțiuni scrise pentru tratarea unor evenimente specifice sfârşitului de program (Unload, Terminate etc.).

Fişierele deschise prin Open sunt închise şi toate variabilele sunt eliberate. Obiectele create din modulele clasă sunt distruse, iar referințele din alte aplicații la asemenea obiecte sunt invalidate. Memoria este eliberată.

Exit

Prin instrucțiunea Exit, sub una din multiplele ei forme, se întrerupe o ramură de execuție (cum ar fi o procedură, o structură iterativă etc.) pentru a se continua nivelul apelant. Sintaxa este

Exit Do

Exit For

Exit Function

Exit Property

Exit Sub

şi efectele sunt prezentate la structurile respective. Nu trebuie confundată cu instrucțiunea End.

Stop

Efectul instrucțiunii este dependent de modul de execuției a programului. Dacă se execută varianta compilată a programului (fişierul .exe) atunci instrucțiunea este similară instrucțiunii End (suspendă execuția şi închide fişierele deschise). Dacă execuția este din mediul VBA, atunci se suspendă execuția programului, dar nu se închid fişierele deschise şi nu se şterge valoarea variabilelor. Execuția poate fi reluată din punctul de suspendare.

Stop

Instrucțiunea este similară introducerii unui punct de oprire (Breakpoint) în codul sursă.

Structuri iterative (Do...Loop, For...Next, For Each...Next, While...Wend, With)

Prin intermediul construcțiilor de tip bloc prezentate în această secțiune se poate repeta, în mod controlat, un grup de instrucțiuni. În cazul unui număr nedefinit de repetiții, condiția de oprire poate fi testată la începutul sau la sfârşitul unui ciclu, prin alegerea structurii adecvate.

Do…Loop

Se vor utiliza structuri Do…Loop pentru a executa un grup de instrucțiuni de un număr de ori nedefinit aprioric. Dacă se cunoaşte numărul de cicluri, se va utiliza structura For…Next.

Înainte de continuare se va testa o condiție (despre care se presupune că poate fi modificată în instrucțiunile executate). Diferitele variante posibile pentru Do…Loop diferă după momentul evaluării condiției şi decizia luată.

Do [{While | Until} condition**]**

[statements**]**

[Exit Do]

[statements**]**

Loop

sau

Do

[statements**]**

[Exit Do]

[statements**]**

Loop [{While | Until} condition**]**

unde

condition este o expresie care valoare de adevăr True sau False. O condiție care este Null se consideră False.

statements sunt instrucțiounile care se repetă atâta timp (while) sau până când (until) condiția devine True.

Dacă decizia este de a nu continua ciclarea, atunci se va executa prima instrucțiune care urmează întregii structuri (deci de după linia care începe cu Loop).

Se poate abandona ciclarea oriunde în corpul structurii prin utilizarea comenzii Exit Do (cu această sintaxă). Dacă apare o comandă Exit Do se poate omite chiar şi condiția din enunț întrucât execuția se va termina prin această decizie.

Structurile Do pot fi inserate (dar complet) unele în altele. O terminare (prin orice metodă) a unei bucle transferă controlul la nivelul Do imediat superior.

Execuția structurilor este explicată în tabelul următor

Do While…Loop	Testează condiția la începutul buclei, execută bucla numai dacă rezultatul este True şi continuă astfel până când o nouă evaluare produce False.
Do Until…Loop	Testează condiția la începutul buclei, execută bucla numai dacă rezultatul este False şi continuă astfel până când o nouă evaluare produce True.
Do…Loop While	Se execută întotdeauna bucla o dată, se testează condiția la sfârşitul buclei şi se repetă bucla atât timp cât condiția este True. Oprirea este pe condiție falsă.
Do…Loop Until	Se execută întotdeauna bucla o dată, se testează condiția la sfârşitul buclei şi se repetă bucla atât timp cât condiția este False. Oprirea este pe condiție adevărată.

For…Next

Atunci când se cunoaşte numărul de repetări ale unui bloc de instrucțiuni, se va folosi structura For…Next. Structura utilizează o variabilă contor, a cărei valoare se modifică la fiecare ciclu, oprirea fiind atunci când se atinge o valoare specificată. Sintaxa este:

For counter = start **To** end **[Step** step**]**

[statements**]**

[Exit For]

[statements**]**

Next [counter**]**

unde

counter este variabila contor (numără repetările), de tip numeric. Nu poate fi de tip Boolean sau element de tablou.

start este valoarea inițială a contorului.

end este valoarea finală a contorului.

step este cantitatea care se adună la contor la fiecare pas. În cazul în care nu se specifică este implicit 1. Poate fi şi negativă.

statements sunt instrucțiunile care se repetă. Dacă nu se specifică, atunci singura acțiune este cea de modificare a contorului de un număr specificat de ori.

Acțiunea este dictată de pasul de incrementare şi relația dintre valoarea inițială şi cea finală.

Instrucțiunile din corpul structurii se execută dacă

counter <= end pentru step >= 0 sau

counter >= end pentru step < 0.

După ce toate instrucțiunile s-au executat, valoarea step este adăugată la valoarea contorului şi instrucțiunile se execută din nou după acelaşi test ca şi prima dată, sau bucla For…Next este terminată şi se execută prima instrucțiune de după linia Next.

Specificarea numelui contorului în linia Next poate clarifica textul sursă, mai ales în cazul când există structuri For…Next îmbricate.

Corpul unei bucle For…Next poate include (complet) o altă structură For…Next. În asemenea situații, structurile îmbricate trebuie să aibă variabile contor diferite.

Instrucțiunile Exit For pot fi plasate oriunde în corpul unei bucle şi provoacă abandonarea ciclării. Controlul execuției se transferă la prima instrucțiune de după linia Next.

For Each…Next

Similară structurii For…Next, structura For Each…Next repetă un grup de instrucțiuni pentru fiecare element dintr-o colecție de obiecte sau dintr-un tablou (cu excepția celor de un tip utilizator). Este utilă atunci când nu se cunoaşte numărul de elemente sau dacă se modifică, în timpul execuției, conținutul colecției.

Sintaxa este:

For Each element **In** group

[statements**]**

[Exit For]

[statements**]**

Next [element**]**

unde

element este variabila utilizată pentru parcurgerea elementelor. Dacă se parcurge o colecție de obiecte, atunci *element* poate fi Variant, o variabilă generică de tip Object, sau o variabilă obiect specifică pentru biblioteca de obiecte referită. Pentru parcurgerea unui tablou, *element* poate fi doar o variabilă de tip Variant.

group este numele colecției de obiecte sau al tabloului.

statements este grupul de istrucțiuni executate pentru fiecare element.

Execuția unei structuri For Each…Next este

1. Se defineşte *element* ca numind primul element din grup (dacă nu există nici un element, se transferă controlul la prima instrucțiune de după Next – se părăseşte bucla fără executarea instrucțiunilor).

Se execută instrucțiunile din corpul buclei For.

Se testează dacă element este ultimul element din grup. Dacă răspunsul este afirmatif, se părăseşte bucla.

2. Se defineşte *element* ca numind următorul element din grup.

Se repetă paşii 2 până la 4.

Instrucțiunile Exit For sunt explicate la For…Next.

Buclele ForEach...Next pot fi îmbricate cu condiția ca elementele utilizate la iterare să fie diferite.

Observație. Pentru ştergerea tuturor obiectelor dintr-o colecție se va utiliza For…Next şi nu For Each…Next. Se va utiliza ca număr de obiecte *colecție.**Count***.

While…Wend

Execută un grup de instrucțiuni atât timp cât este adevărată o condiție. Sintaxa

While condition

[statements**]**

Wend

Este recomandat să se utilizeze o structură Do…Loop în locul acestei structuri.

With

Programarea orientată pe obiecte produce, datorită calificărilor succesive, construcţii foarte complexe atunci când se numesc proprietăţile unui obiect. În cazul modificărilor succesive ale mai multor proprietăţi ale aceluiaşi obiect, repetarea zonei de calificare poate produce erori de scriere şi conduce la un text greu de citit. Codul este simplificat prin utilizarea structurii With…End With. O asemenea structură execută o serie de instrucţiuni pentru un obiect sau pentru o variabilă de tip utilizator. Sintaxa este:

With object

[statements**]**

End With

unde

object este numele unui obiect sau a unui tip definit de utilizator

statements sunt instrucţiunile care se execută pentru entitatea precizată.

Permiţând omiterea recalificărilor din referinţele la obiectul precizat, orice construcţie de tipul

".nume" este interpretată în instrucţiunile structurii drept "*object*.nume".

Într-un bloc With nu se poate schimba obiectul procesat.

La plasarea unui bloc With în interiorul altui bloc With, obiectul extern este mascat complet, deci calificările eventuale la acest obiect vor fi efectuate.

Nu se recomandă saltul în şi dintr-un bloc With.

Structuri de decizie (If…Then…Else, Select Case)

Ramificarea firului execuţiei după rezultatul verificării unei condiţii este o necesitate frecventă în orice implementare.

Pe lângă structurile prezentate, se pot utiliza trei funcţii care realizează alegeri în mod liniarizat (pe o linie de cod): Choose(), Iif(), Switch().

If…Then…Else

O asemenea structură, întâlnită de altfel în toate limbajele de programare, execută un grup de instrucţiuni ca răspuns la îndeplinirea unei condiţii (compusă sau nu din mai multe condiţii testate secvenţial). Sintaxa permite o mare varietate de forme:

If condition **Then [**statements**] [Else** elsestatements**]**

sau

If condition **Then**

[statements**]**

[ElseIf condition-n **Then**

[elseifstatements**]** ...

[Else

[elsestatements**]]**

End If

unde

condition are una din formele: expresie numerică sau şir care se poate evalua True sau False (Null este interpretat False);
expresie de forma **TypeOf** *objectname* **Is** *objecttype*, evaluată True dacă objectname este de tipul obiect specificat în objecttype.

statements, elsestatements, elseifstatements sunt blocurile de instrucţiuni executate atunci când condiţiile corespunzătoare sunt True.

La utilizarea primei forme, fără clauza Else, este posibil să se scrie mai multe instrucţiuni, separate de ":", pe aceeaşi linie.

Verificarea condiţiilor implică evaluarea tuturor subexpresiilor, chiar dacă prin jocul operanzilor şi operatorilor rezultatul poate fi precizat mai înainte (de exemplu OR cu primul operand True).

Select Case

Instrucţiunea Select Case se poate utiliza în locul unor instrucţiuni ElseIf multiple (dintr-o structură If…Then…ElseIf) atunci când se compară aceeaşi expresie cu mai multe valori, diferite între ele. Instrucţiunea Select Case furnizează, prin urmare, un sistem de luare a deciziilor similar instrucţiunii If…Then…ElseIf. Totuşi, Select Case produce un un cod mai eficient şi mai inteligibil. Sintaxa este:

Select Case testexpression

[Case expressionlist-n

[statements-n**]] ...**

[Case Else

[elsestatements**]]**

End Select

unde

testexpression este o expresie numerică sau şir.

expressionlist-n este lista, separată prin virgule, a uneia sau mai multe expresii de forma:

- **expression**.
- **expression To expression**. Cuvântul **To** introduce un interval de valori, valoarea minimă fiind prima specificată.
- **Is comparisonoperator expression**. Se va utiliza **Is** cu operatori de comparare (exceptând **Is** şi **Like**) pentru a specifica un domeniu de valori.

statements-n reprezintă una sau mai multe instrucţiuni care se vor executa dacă *testexpression* este egală cu un element din *expressionlist-n*.

elsestatements reprezintă una sau mai multe instrucţiuni care se vor executa dacă *testexpression* nu este egală cu nici un element din listele liniilor Case.

Dacă testexpression se potriveşte cu un element dintr-o listă Case, se vor executa instrucţiunile care urmează această clauză Case până la următoarea clauză Case, sau până la End Select. Control execuţiei trece apoi la instrucţiunea care urmează liniei finale End Select. Rezultă că dacă testexpression se regăseşte în mai multe liste, doar prima potrivire este considerată.

Clauza Case Else are semnificaţia uzuală "altfel, în rest, în caz contrar etc.", adică introduce instrucţiunile care se execută atunci când expresia de test nu se potriveşte nici unui element din listele clauzelor Else. Dacă aceasta este situaţia şi nu este specificată o clauză Case Else, atunci execuţia urmează cu prima instrucţiune de după End Select.

Instrucţiunile Select Case pot fi scufundate unele în altele, structurile interioare fiind complete (fiecare structură are End Select propriu, includerea este completă).

Apeluri de proceduri şi programe

În această secţiune se prezintă doar funcţia Shell(), deoarece despre proceduri şi apelul lor s-a discutat în capitolul 1.

Funcţia Shell()

Execută un program executabil şi returnează un Variant(Double) reprezentând ID-ul de task al programului în caz de succes; în caz contrar returnează zero. Sintaxa este

Shell(pathname**[**,windowstyle**])**

unde

pathname este Variant (String). Conţine numele programului care se execută, argumentele necesare şi poate da calea completă (dacă este nevoie).

windowstyle este Variant (Integer) şi precizează stilul ferestrei în care se va executa programul (implicit este minimizat, cu focus).

Valorile posibile pentru argumentul windowstyle sunt

Constanta numită	Valoarea	Semnificaţia
VbHide	0	Fereastra este ascunsă iar focus-ul este pe fereastra ascunsă.
VbNormalFocus	1	Fereastra are focus-ul şi este dimensionată şi poziţionată normal.
VbMinimizedFocus	2	Fereastra este afişată ca o icoană (minimizată) dar are focus-ul.
VbMaximizedFocus	3	Fereastră maximizată, cu focus.
VbNormalNoFocus	4	Fereastra este normală (restaurată la mărimea şi poziţia cea mai recentă) dar nu are focus-ul. Fereastra activă curentă îşi păstrează focus-ul.
VbMinimizedNoFocus	6	Fereastră minimizată, fără focus. Fereastra activă curentă îşi păstrează focus-ul.

Dacă funcţia Shell nu poate porni programul specificat se va semnala eroare. Programul pornit prin Shell se execută asincron, deci nu există certitudinea că acest program se termină înainte de execuţia instrucţiunilor care urmează liniei Shell.

Operaţiuni de intrare-ieşire

În categoria operaţiunilor de I/O se pot deosebi

schimbul de informaţii cu utilizatorul: acesta se poate desfăşura prin intermediul unor formulare (forms) predefinite (InputBox, MsgBox) sau prin intermediul unor formulare definite de dezvoltatorul proiectului VBA.

schimbul de informaţii cu fişiere şi/sau baze de date: acesta se realizează prin intermediul unor instrucţiuni dedicate acestor operaţii.

Dialogul standard cu utilizatorul

În general, utilizatorul răspunde la apariţia unei boxe de dialog prin acţionarea butonului adecvat răspunsului său şi/sau prin înscrierea unei informaţii într-o zonă dedicată acestui scop. Informaţia înscrisă este transferată programului ca valoare a funcţiei care iniţiază dialogul grafic.

Funcţia InputBox

Apelul funcţiei InputBox afişează o boxă de dialog care conţine un mesaj, două butoane (OK şi Cancel) şi o zonă text în care se poate tasta un răspuns (de tip String, chiar dacă se transmite o valoare numerică). Se aşteaptă ca utilizatorul să introducă un text în zona rezervată sau să acţioneze un buton. Textul introdus este transmis ca valoare a funcţiei la acţionarea butonului OK (sau Enter), iar acţionarea butonului Cancel (ca şi închiderea dialogului ca fereastră) transmite un şir de lungime zero (indiferent de valoarea zonei text. Sintaxa este

InputBox(prompt[, title] [, default] [, xpos] [, ypos] [, helpfile, context])

unde

prompt este expresia String cu mesajul afişat în dialog (max. 1024 caractere). Mesajul poate fi aranjat pe mai multe linii prin combinaţii de caractere Chr(13) – carriage return, Chr(10) – linefeed, Chr(13)&Chr(10) – CR+LF.

title este expresia de tip String cu titlul ferestrei dialogului. Dacă este omis se va folosi titlul aplicației.

default este expresia de tip String, opțională, conținând textul afişat inițial în zona text rezervată utilizatorului. Textul este, în lipsa modificării lui, retransmis ca răspuns (acționând butonul OK).

xpos este expresia numerică specificând poziția orizontală a dialogului (în twips, de la latura stânga a ecranului). În lipsa argumentului, boxa de dialog este centrată orizontal.

ypos este expresia numerică specificând poziția verticală a dialogului (în twips, de la latura de sus a ecranului). În lipsa argumentului, boxa de dialog este poziționată la aproximativ o treime de ecran.

helpfile este expresia de tip String care identifică fişierul Help utilizat.Dacă este indicat *helpfile,* trebuie să fie precizat şi *context*. Textul de ajutor poate fi văzut prin tasta F1, anumite aplicații afişează şi un buton Help.

context Expresie numerică cu numărul de context Help al intrării corespunzătoare dialogului afişat. Apare obligatoriu împreună cu *helpfile*.

Dacă valoarea funcției nu este atribuită (sau utilizată într-o altă expresie), este evident că se pierde, fără semnalarea vreunei erori.

Funcția MsgBox

Un dialog mai simplu decât InputBox este realizat prin forma predefinită afişată de funcția MsgBox. Se afişează un mesaj într-o boxă de dialog şi se aşteaptă ca utilizatorul să acționeze unul dintre butoanele existente (numărul şi tipul lor lor este fixat la proiectarea aplicației). Funcția returnează un întreg indicând care buton a fost acționat. Sintaxa este

[intvariable=]**MsgBox**(prompt[, buttons] [, title] [, helpfile, context])

unde

prompt este textul mesajului (Vezi InputBox)

buttons este o expresie numerică egalăcu suma valorilor care specifică numărul, tipul şi atributele butoanelor. Aici se fixează şi modalitatea ferestrei (Vezi constantele predefinite din tabelul care urmează). Valoarea implicită este 0.

title,helpfile, context identice cu argumentele similare descrise la InputBox.

Constantele recomandate pentru formarea argumentului *buttons* sunt

Constanta	Valoare	Descriere
VbOKOnly	0	Numai butonul OK
VbOKCancel	1	OK şi Cancel
VbAbortRetryIgnore	2	Abort, Retry şi Ignore
VbYesNoCancel	3	Yes, No şi Cancel
VbYesNo	4	Yes şi No
VbRetryCancel	5	Retry şi Cancel
VbCritical	16	Icoana de Critical Message
VbQuestion	32	Icoana de Warning Query
VbExclamation	48	Icoana Warning Message
VbInformation	64	Icoana de Information Message

VbDefaultButton1	0	Primul buton este cel implicit (echivalent cu Enter)
VbDefaultButton2	256	Al doilea buton este cel implicit
VbDefaultButton3	512	Al treilea buton este cel implicit
VbDefaultButton4	768	Al patrulea buton este cel implicit
VbApplicationModal	0	Application modal: aplicatia curentă este oprită până când răspunde utilizatorul
VbSystemModal	4096	System modal: toate aplicațiile sunt oprite până când utilizatorul răspunde la dialog

Valorile 0-5 descriu butoanele, 16,32,48 şi 64 descriu stilul icoanei afişate, 0, 256 şi 512 determină butonul implicit, iar ultimul grup (0 şi 4096) determină modalitatea boxei de dialog. La formarea argumentului Buttons se va adună doar câte un număr din fiecare grup.

Pentru a utiliza valoarea returnată de funcție, aceasta trebuie inclusă într-o expresie (eventual atribuită unei variabile întregi).

Valorile returnate de funcție şi care pot fi testate, în expresii logice, pentru a alege ramura de prelucrare dorită de utilizator sunt

Constanta	Valoare	Descriere
VbOK	1	OK
VbCancel	2	Cancel
VbAbort	3	Abort
VbRetry	4	Retry
VbIgnore	5	Ignore
VbYes	6	Yes
VbNo	7	No

Acționarea tastei Esc este echivalentă cu acționarea butonului Cancel (dacă acesta este prezent). Dacă în dialog este prezent butonul Help, acționarea lui nu termină dialogul.

Utilizarea fişierelor

Procesările tipice programate în VBA prelucrează informații din două mari categorii de fişiere:

fişiere ale aplicațiilor server (.doc în Word, .xls în Excel etc.)

fişiere utilizator (create şi/sau gestionate de proiect pentru date de intrare, temporare sau de ieşire).

> *Accesarea directă a fişierelor din prima categorie (fără apelul aplicației server specifice) poate produce coruperea fişierului, astfel încât nu mai este recunoscut de aplicația mamă. Prelucrarea acestor fişiere trebuie să fie executată în aplicațiile care le-au creat.*

Pentru lucrul cu un fişier utilizator (în continuare prin fişier se va înţelege, fără alte precizări, un fişier utilizator) acesta trebuie mai întâi deschis (instrucţiunea Open), operaţiunea producând şi crearea fişierului în cazul unui fişier inexistent (nou). După utilizare fişierul trebuie să fie închis (operaţiune efectuată, la terminarea normală a programului, în mod automat).

Un fişier are ataşat un număr de identificare, unic pentru un proces. Identificare fişierului se poate efectua, în program, prin numele său sau prin numărul ataşat. Numărul poate fi în domeniul 1–255 pentru fişierele proprii aplicaţiei şi în domeniul 256–511 pentru fişiere accesibile din alte aplicaţii. Un număr neutilizat (liber) poate fi furnizat de apelul la funcţia FreeFile().

Există trei moduri de acces la înregistrările unui fişier, acces definit la deschiderea acestuia.

acces secvenţial (modurile Input, Output şi Append), utilizat de regulă pentru scrierea fişierelor text (rapoarte, jurnale etc.);

acces raandom (aleator) (modul Random), în cazul când este necesar să se scrie şi să se citească înregistrările într-o ordine nedefinită, operaţiunile de intrare/ieşire fiind amalgamate între ele;

acces binar (modul Binary), utilizat la citirea/scrierea fişierelor byte cu byte (de exemplu fişiere bitmap).

Un fişier deschis cu un mod de acces trebuie exploatat în acest mod până când este închis şi deschis în alt mod (dacă structura lui permite aşa ceva).

Instrucţiunile tipice pentru accesul la informaţiile dintr-un fişier sunt

Modul de acces	Scriere	Citire
Secvenţial	Print #, Write #	Input #
Random	Put	Get
Binar	Put	Get

Deoarece gestionarea fişierelor nu se rezumă doar la scriere/citire, în tabelul următor este un rezumat al principalelor operaţiuni pe care le suportă fişierele, cu instrucţiunile care facilitează respectiva acţiune.

Acţiune	Instrucţiuni
Citire	**Get, Input, Input #, Line Input #**
Controlul ieşirilor	Format, Print, **Print #, Spc, Tab,** Width #
Copierea unui fişier	FileCopy
Creare, acces	**Open**
Fixarea atributelor	FileAttr, GetAttr, SetAttr
Fixarea poziţiei active de citire/scriere	**Seek**
Inchidere	**Close, Reset**
Informaţii despre un fişier	**EOF,** FileAttr, FileDateTime, FileLen, **FreeFile**, GetAttr, **Loc, LOF, Seek**

Lungimea unui fişier	FileLen
Operaţii asupra fişierelor	Dir, Kill, Lock, Unlock, Name
Scriere	**Print #, Put, Write #**

Doar instrucţiunile şi funcţiile des utilizate sunt prezentate în continuare, pentru celelalte se va studia intrarea corespunzătoare din Help (în mediul VBA).

Open

Deschide un fişier în sensul că rezervă o zonă tampon (buffer) pentru fişier şi determină modul de acces utilizat. Nu se pot efectua instrucţiuni de I/O pe un fişier dacă acesta nu este deschis în prealabil. Sintaxa:

Open pathname **For** mode **[Access** access**] [**lock**] As [#]**filenumber **[Len=**reclength**]**

unde

pathname expresie String care specifică numele fişierului (poate include întreaga cale unitate, directoare etc., după regulile uzuale);

mode cuvânt cheie care specifică modul de acces la fişier: *Append, Binary, Input, Output* sau *Random*; dacă nu se specifică nimic se va considera acces *Random*;

access clauză opţională specificând operaţiunile I/O permise pentru fişier: *Read, Write sau Read Write;*

lock clauză opţională specificând operaţiile asupra fişierului permise altor procese care se execută (când fişierul este deschis): *Shared, Lock Read, Lock Write şi Lock Read Write.*

filenumber numărul de fişier pentru fişierul deschis (între 1 şi 511, vezi observaţia din partea introductivă); funcţia FreeFile furnizează următorul număr disponibil;

reclength număr (<=32,767) exprimând, în octeţi, lungimea înregistrării (la aceesul *Random*) sau lungimea bufferului (la accesul secvenţial); la accesul Binary, clauza este ignorată.

Dacă fişierul indicat nu există, atunci este creat un fişier cu acest nume în cazurile care presupun o ieşire în fişier: modurile Append, Binary, Output sau Random.

Un fişier poate fi deschis în mai multe moduri simultan (prin instrucţiuni Open distincte pentru fiecare mod, cu numere diferite) dacă există compatibilitate între moduri: Binary, Input şi Random permit acest lucru, Append, Output nu permit (fişierul trebuie mai întâi închis şi abia apoi deschis într-un asemenea mod).

FreeFile

Funcţia returnează un întreg reprezentând următorul număr de fişier disponibil.

FreeFile[(rangenumber)**]**

rangenumber este un Variant care specifică domeniul din care se solicită un număr liber de fişier: 0 (valoarea implicită) returnează un număr în 1–255 (pentru fişiere proprii, 1 returnează un număr în 256–511 (pentru fişiere accesate din alte aplicaţii).

Get

Citeşte date dintr-un fişier deschis şi le transferă într-o variabilă. Datele citite cu Get sunt, în general, scrise în fişier cu comanda Put.

Get [#]filenumber, **[**recnumber**]**, varname

unde

filenumber este numărul fişierului de unde se citesc date (fişierul trebuie să fie deschis)

recnumber număr opţional în format Variant (Long), reprezintă numărul înregistrării (modul *Random*) sau numărul octetului (modul Binary) de unde începe citirea. Prima poziţie este 1.

varname numele variabilei unde se transferă informaţia.

Dacă nu se specifică numărul înregistrării se va citi din poziţia activă de după ultima instrucţiune Get, Put sau Seek. Argumentul lipsă este indicat prin virgule: Get #4,,FileBuffer.

Pentru observaţiile privind acţiunea instrucţiunii Get, separat pentru modul Random şi Binary, se va studia Help - Get. Observaţiile sunt utile atunci când se operează, în special, cu tipurile Variant şi cu tablouri.

Put

Scrie valoarea unei variabile date într-un fişier deschis în prealabil. Datele scrise cu Put sunt, în general, citite din fişier cu Get.

Put [#]filenumber, **[**recnumber**]**, varname

unde

filenumber este numărul fişierului unde se scriu datele (fişierul trebuie să fie deschis);

recnumber număr opţional în format Variant (Long), reprezintă numărul înregistrării (modul *Random*) sau numărul octetului (modul Binary) unde începe scrierea. Prima poziţie este 1.

varname numele variabilei a cărei valoare se scrie în fişier.

Dacă nu se specifică numărul înregistrării se va scrie în poziţia activă de după ultima instrucţiune Get, Put sau Seek. Argumentul lipsă este indicat prin virgule: Put #4,,FileBuffer.

Pentru observaţiile privind acţiunea instrucţiunii Put, separat pentru modul Random şi Binary, se va studia Help - Put. Observaţiile sunt utile atunci când se operează, în special, cu tipurile Variant şi cu tablouri.

Input

Citeşte date dintr-un fişier secvenţial şi le transferă în variabilele specificate.

Instrucţiunea se va utiliza doar cu fişierele deschise în modul Input sau Binary, datele citite cu Input # sunt scrise, de regulă, cu Write #.

Input #filenumber, varlist

unde

filenumber numărul fişierului (deschis în prealabil);

varlist listă de variabile, delimitate de virgule, pentru care se citesc valorile din fişier. Nu se pot include nume de tablouri sau variabile Object, dar se acceptă elemente de tablou şi variabiel de tipuri utilizator.

Pentru situaţiile uzuale (tipuri numerice sau String standarde) asignarea valorilor se efectuează fără modificări. Pentru alte situaţii:

Informaţia citită	**Valoarea asignată**
Virgulă sau linie goală	Empty
#NULL#	Null
#TRUE# sau #FALSE#	True sau False
#yyyy-mm-dd hh:mm:ss#	Data şi/sau timpul reprezentat de expresie
#ERROR errornumber#	errornumber (variabila este un Variant considerat drept eroare)

Ghilimelele duble (" ") sunt ignorate în şirul de intrare.

Pentru o citire corectă, datele din fişier trebuie să apară în aceeaşi ordine şi de acelaşi tip cu variabilele din listă. O variabilă numerică primeşte valoarea 0 dacă intrarea corespunzătoare nu este numerică. Atingerea sfârşitului de fişier când operaţiunea de citire nu este încheiată, provoacă eroare.

Întrucât utilizarea fişierelor este, în mod uzual, aceea de memorare controlată a unor informaţii (şi nu aceea de a descifra informaţii scrise într-o structură necunoscută), se recomandă scrierea cu Write # în cazul utilizării ulterioare a comenzii Input #.

Funcţia Input()

Citeşte şi returnează un şir de caractere citite dintr-un fişier deschis în mod Input sau Binary. Datele citite prin această funcţie sunt scrise, de regulă, prin Print # sau Put.

Input(number, **[#]**filenumber**)**

unde

number orice expresie numerică specificând numărul de caractere care se citesc.

filenumber număr de fişier (deschis).

Spre deosebire de instrucţiunea Input #, funcţia Input returnează toate caracterele citite (inclusiv virgule, CR, LF, ghilimele şi spaţii de început).

Pentru fişierele deschise pentru acces Binary, încercarea de a citi prin funcţia Input până când EOF returnează True generează eroare (procedeul este valid pentru citirea din fişiere binare cu Get). Se vor utiliza funcţiile LOF and Loc pentru detectarea sfârşitului de fişier.

Observaţie. Pentru date pe octeţi din fişiere text se va utiliza funcţia InputB, cu o sintaxă similară, unde number specifică numărul de octeţi de returnat (în loc de numărul de caractere). A se vedea şi Help – Returning Strings from Functions.

Line Input

Citeşte o singură linie dintr-un fişier secvenţial (deschis) şi asignează şirul obţinut unei variabile de tip String. O linie este considerată terminată la întâlnirea caracterului CR (Chr(13)) sau a combinaţiei CR+LF (Chr(13)&Chr(10)). Caracterele CR şi/sau LF nu sunt adăugate şirului asignat (se poate considera că secvenţa lor a fost sărită).

Line Input #filenumber, varname

unde

filenumber numărul ataşat fişierului (deschis),

varname nume de variabilă String sau Variant.

Datele citite cu Line Input # sunt, de regulă, scrise cu Print #.

Write

Scrie o înregistrare într-un fişier secvenţial. Datele scrise prin Write # sunt citite, de regulă, cu Input #. Utilizarea scrierii cu Write # asigură o delimitare corectă a fiecărui câmp scris, ceea ce permite regăsirea corectă (fără alte artificii) a informaţiilor la citirea cu Input #. În acelaşi timp, informaţiile sunt regăsite corect indiferent de configurările locale.

Sintaxa este

Write #filenumber, **[**outputlist**]**

unde

filenumber numărul ataşat fişierului (deschis în prealabil),

outputlist o listă de expresii numerice sau şir, separate prin virgule, spaţii sau punct-virgulă, ale căror valori se scriu în fişier.

Specificarea unei virgule după filenumber fără outputlist produce o linie goala în fişier.

Sunt respectate urtmătoarele reguli de scriere:

datele numerice sunt scrise cu punct ca separator zecimal (indiferent de setările locale);

datele Boolean sunt scrise ca #TRUE# sau #FALSE#, nefiind traduse după setările locale;

datele calendaristice şi timpul sunt scrise potrivit formatului de dată universală; dacă o componentă este omisă (sau este zero), se scrie doar partea indicată;

Null se scrie drept #NULL#, iar Empty nu produce nimic în ieşire;

Date de tip Error apare #ERROR errorcode#.

Instrucțiunea Write # inserează între virgule între elementele scrise în fişier, ca şi ghilimele în jurul şirurilor de caractere (nu este prin urmare nevoie ca utilizatorul să introducă separatori pentru claritate). După ce toate valorile au fost scrise, se inserează automat o combinație CR+LF, astfel încât următoarea scriere va fi pe un rând nou.

Print

Scrie într-un fişier secvențial date formatate ca pe ecran (display-formatted). Prin urmare, cu excepțiile specificate în continuare, setările locale sunt respectate. Datele scrise cu Print # sunt, de regulă, citite cu Line Input # sau cu Input.

Sintaxa este

Print #filenumber, **[**outputlist**]**

unde

filenumber numărul ataşat fişierului (deschis în prealabil),

outputlist listă de expresii formatate ale căror valori sunt tipărite. Elementele se separă prin virgule, spații sau punct şi virgulă.

Un element al listei de ieşire este de forma

[{Spc(n) **| Tab[**(n)**]}] [**expression**] [**charpos**]**

unde

Spc(n) inserează n spații în ieşire

Tab(n) poziționează punctul de inserție (începutul zonei de scriere) la o coloană indicată absolut de n. Utilizând doar Tab se trece la următoarea zonă de ieşire.

expression expresia a cărei valoare se tipăreşte (numerică sau String).

charpos Specifică poziția punctului de inserție pentru următorul caracter care va fi tipărit, potrivit tabelului care urmează. Dacă nu se specifică, următoarea tipărire va fi pe rândul următor.

charpos	**Locul punctului de inserție**
;	Imediat după ultimul caracter tipărit
Tab(n)	Coloana cu numărul n
Tab	Începutul următoarei zone de tipărire

Dacă se omite *outputlist* dar se include un separator după *filenumber*, se va insera o linie goală în fişier.

Datele logice sunt scrise drept True, False (fără traduceri locale).

Datele calendaristice sunt scrise potrivit setării locale pentru format scurt.

Empty nu produce nimic, Null este scris Null, iar Error este scris ca Error errcode (fără traduceri locale).

Informațiile numerice scrise sunt după configurările locale (separator zecimal).

Pentru o interpretare corectă, utilizatorul trebuie să separe valorile afişate prin formatări adecvate.

Pentru afişari în fereastra Immediate a mediului VBA, se va vedea şi metoda Print (vezi Help – Print Method).

Close

Închide unul sau mai multe fisiere deschise prin Open pentru instrucțiuni de I/O. Prin acestă operațiune se rupe legătura între fişiere şi numerele ataşate şi se eliberează zonele tampon rezervate. Pentru un fişier închis nu se mai pot executa operațiuni I/O (până la o nouă deschidere).

Sintaxa este

Close [filenumberlist**]**

unde

filenumberlist este lista de numere ataşate fişierelor care se închid, de forma [[#]filenumber] [, [#]filenumber] …; dacă lista nu este prezentă, atunci se vor închide toate fişierele care sunt deschise.

Înainte de închidere, în fişierele deschise pentru Output sau Append se scriu zonele buffer nescrise încă.

Reset

Închide toate fişierele deschise prin instrucţiuni Open. Sintaxa este

Reset

Înaine de închidere se scriu în fişiere toate bufferele nescrise încă.

Seek

Stabileşte poziţia (înregistrării sau octetului) într-un fişier unde se va efectua următoarea operaţiune de intrare/ieşire, fişierul fiind deschis prin Open (vezi şi funcţia Seek). Sintaxa

Seek [#]filenumber, position

unde

filenumber numărul ataşat fişierului.

position număr între 1 şi 2,147,483,647, inclusiv, care indică locul următoarei operaţii I/O.

Numerele înregistrărilor specificate în instrucţiunile Get şi Put au priorirtate în raport cu poziţia fixată prin Seek (are loc o repoziţionare).

Dacă operaţiunea Seek indică o poziţie după sfârşitul fişierului, următoarea operaţiune de scriere (fără repoziţionare) extinde fişierul.

Poziţia indicată nu poate fi zero sau negativă.

Funcţia Seek

Returnează, ca un întreg Long, poziţia curentă I/O dintr-un fişier specificat. Fişierul trebuie să fie în prealabil deschis.

Seek(filenumber**)**

filenumber este un număr de fişier.

Valoarea returnată este între 1 şi 2,147,483,647 (echivalent cu 2^31 – 1), inclusiv (vezi şi setarea poziţiei prin instrucţiunea Seek) şi are semnificaţia din următorul tabel.

Modul de acces	Valoarea returnată
Random	Numărul următoarei înregistrări (care va fi citită sau scrisă)
Binary, Output, Append, Input	Poziţia octetului (numerotat de la 1) la care va avea loc următoarea operaţiune I/O

EOF

Returnează un întreg cu valoarea logică True (-1) atunci când se atinge sfârşitul unui fişier deschis pentru citire (Random, Binary sau Input). Pentru fişierele dschise în ieşire funcţia generează mereu True.

EOF(filenumber**)**

filenumber este un întreg conţinând numărul fişierului testat.

Utilizarea uzuală este

```
Do While Not EOF(filenum)
        … (instrucţiuni, inclusiv citire din fişierul filenum)
```

```
Loop
```

Pentru acces secvențial (Input) se întoarce False până când se atinge sfârşitul de fişier, pentru fişierele Random sau Binary se returnează False până când ultima instrucțiune Get executată nu a putut citi o înregistrare întreagă.

Citirea cu Input dintr-un fişier deschis Binary produce eroare la utilizarea mecanismului general (până când EOF () este true): se va utiliza citirea cu Get sau citirea cu Input împreună cu funcțiile LOF sau Loc.

Loc

Returnează, ca Long, poziția curentă de citire/scriere într-un fişier deschis.

Loc(filenumber**)**

filenumber este numărul ataşat fişierului.

Valoarea funcției depinde de modul de acces

Mod	Valoarea returnată
Random	Numărul ultimei înregistrări scrise sau citite
Sequential	Poziția curentă împărțită la 128. (Se spune că această informație nu este niciodată utilă sau utilizată)
Binary	Poziția ultimului octet citit sau scris.

Funcția Loc este utilizată, împreună cu funcția LOF, la testarea sfârşitului de fişier la citiri Binary (schema generală este dată la LOF).

LOF

Returnează un Long care reprezintă, în octeți, mărimea unui fişier deschis prin Open. Pentru fişierele nedeschise se poate utiliza, în acelaşi scop, funcția FileLen().

LOF(filenumber**)**

filenumber este numărul ataşat fişierului.

Utilizarea acestei funcții, împreună cu funcția Loc, pentru determinarea sfârşitului de fişier (similar cu EOF) accesat Binary este după schema generală:

```
Open filename For Binary As filenum
Do While CurrentLocation < LOF (filenumber)
    ... (citire din fişierul filenum )
    CurrentLocation = Loc (filenumber)
    ...
Loop
```

Modele de obiecte

Aproape toate acțiunile programate în VB implică manevrarea programatică a unor obiecte. Toate aplicațiile din Microsoft Office sunt alcătuite din componente formate din obiecte sau care gestionează obiecte.

În această secțiune se prezintă principalele concepte din programarea orientată pe obiecte, ca şi uneltele şi tehnicile disponibile pentru a explora şi utiliza obiectele specifice din Office.

Deoarece fiecare aplicație din Office are un model propriu de obiecte, va fi dedicat câte un capitol pentru Word, Excel etc., în care se vor prezenta particularitățile de operare şi obiectele specifice aplicației.

Privire generală

Orice aplicație poate fi gândită ca ansamblul a două lucruri: conținut şi funcționalitate. Conținutul se referă la documentele pe care le conține aplicația, la elementele care compun documentele, la informațiile privind atributele

elementelor. Funcționalitatea se referă la modurile, căile în care se poate lucra cu conținutul aplicației, de exemplu: deschiderea, închiderea documentelor, adăugarea, copierea, formatarea elementelor etc.

Conținutul şi funcționalitatea unei aplicații sunt divizate în unități discrete de conținut şi funcționalitate specifică, numite obiecte. Exemplele uzuale sunt date de foile de calcul Excel, celule ale unei foi de calcul, secțiuni ale unui document Word etc., fiecare având evident un conținut şi o funcționalitate specifică, cele două componente fiind unitar legate între ele. Obiectele unei aplicații sunt ierarhizate în structuri, modelul de obiecte al aplicației..

Obiectul de nivel maxim al unei aplicații este, uzual, obiectul **Application**, care este aplicația însăşi. Obiectul **Application** conține alte obiecte care pot fi accesate numai când obiectul **Application** există (deci când aplicația se execută). De exemplu, obiectul **Application** Excel conține obiecte **Workbook**, după cum obiectul **Application** Word conține obiecte **Document**. Deoarece obiectul **Document** depinde de existența obiectului **Application** Word, se spune că obiectul **Document** este *copilul* obiectului **Application**; invers, obiectul **Application** se zice *părintele* obiectului **Document**.

Este uzual ca un obiect, care este copil al altui obiect, să aibă, la rândul său, alte obiecte copii. De asemenea, este posibil ca un copil să aibă mai mulți părinți.

Modul în care obiectele, care alcătuiesc o aplicație, sunt aranjate relativ unele față de altele, împreună cu modul în care conținutul şi funcționalitatea sunt divizate prin obiecte este numit *ierarhia de obiecte* sau *modelul de obiecte*. Fiecare aplicație are un model de obiecte propriu, reprezentarea grafică a ierarhiei de obiecte pentru aplicație poate fi văzută în Visual Basic Help din aplicație.

Fiecare obiect din ierarhie are un conținut şi o funcționalitate care se aplică, ambele, atât obiectului însuşi, cât şi tuturor obiectelor descendente din ierarhie. Cu cât obiectul este situat mai sus în ierarhie, cu atât este mai vast domeniul conținutului şi funcționalității sale. Locul unui obiect în model este gândit astfel încât conținutul şi funcționalitatea lui sunt adecvate domeniului său. Se poate gândi şi faptul că, dacă aplicația este divizată în obiecte, fiecare obiect oferă acces la arii specifice de conținut şi funcționalitate.

Afirmațiile care implică obiecte utilizează şi termenii de "conținut în" pentru *copil* şi "conține" pentru *părinte*. Astfel, obiectul **Application** Word *conține* obiecte **Document**, dar obiectul **Selection** este *conținut în* obiectul **Windows** etc.

Proprietăți şi metode

Pentru a avea acces la conținutul şi funcționalitatea unui obiect, pentru început trebuie să se identifice obiectul (subiect discutat în continuare). După identificare, obiectul este accesibil prin intermediul proprietăților şi metodelor sale.

În general, prin proprietate se înțelege un atribut numit al obiectului. Valoarea atributului (proprietății) poate fi modificată (de cele mai multe ori) sau poate fi obținută (ştiută) programatic.

Prin metodă se înțelege o procedură care acționează asupra unui obiect. Pentru a distinge o metodă de o procedură obişnuită (care poate de asemenea să acționeze asupra unui obiect, în general vorbind), trebuie precizat că metodele implementează funcționalitatea obiectului, sunt specifice obiectului căruia i se aplică şi sunt definite o dată cu obiectul (deci la proiectarea aplicației de bază, în cazul obiectelor Office). Orice procedură utilizator acționează asupra obiectului prin intermediul metodelor specifice (aplicabile) acelui obiect.

În general, se utilizează proprietățile pentru a accesa conținutul şi se apelează metodele pentru a realiza funcționalitatea obiectului. Totuşi, această distincție este relativă: există proprietăți care se apropie de metode şi metode care seamănă a fi proprietăți. Atunci când vom discuta despre obiecte definite de utilizator se va vedea că este uşor să se treacă granița dintre metode si proprietăți în proiectarea obiectelor.

Legătura dintre modelul obiectelor şi interfața utilizator

Există două căi prin care utilizatorul poate interacționa cu obiectele aplicației:

manual (utilizând interfața utilizator a aplicației);

programatic (utilizând un limbaj de programare).

În accesul manual se utilizează tastatura, mouse-ul sau cheile directe pentru a naviga către acea parte şi funcție a aplicației care execută ceea ce se doreşte (formatarea unui paragraf, ştergerea unor formule dintr-o celulă, modificarea unui slide etc.).

În accesul programatic, de exemplu în instrucțiuni Visual Basic, se navighează în ierarhia de obiecte pentru a identifica obiectul vizat şi apoi se utilizează proprietatea sau metoda care produce efectul urmărit. De exemplu, prin linia următoare, scrisă într-o procedură,

```
Workbook("Activitate.xls").Worksheets("Vanzari").Range("A5").Value = 100
```

se navighează în caietul Activitate la foaia Vanzari şi se înscrie valoarea 100 în celula A5. Este evident că înscrierea are loc efectiv doar în momentul execuției procedurii.

Deoarece ambele moduri de acces, interfața utilizator a aplicației de bază şi Visual Basic, ajung la acelaşi conținut şi funcționalitate, multe dintre obiectele, proprietățile şi metodele existente în modelele de obiecte Office au aceleaşi denumiri cu elementele din interfața utilizator (denumiri de meniuri, comenzi, acțiuni etc.). Se poate observa, explicabil din punctul de vedere al evoluției către modelele obiectuale, o asemănare globală a modelului de obiecte cu interfața utilizator. Această asemănare este întărită şi de faptul că pentru orice acțiune posibilă prin interfața utilizator există posibilitatea de a scrie cod Visual Basic echivalent (vezi şi discuția cu înregistrarea macro-urilor).

Din exemplul prezentat la accesul din VB, este de reținut importanța cunoaşterii locului ocupat de obiectul procesat în ierarhia de obiecte: pentru a utiliza proprietățile sau metodele lui trebuie identificat corect prin navigarea (calificarea) de la nivelul cel mai de sus până la el. Întregul traseu (cu excepția nivelului Application, care este uneori subînțeles) trebuie specificat ca în exemplul arătat.

Colecții de obiecte

O colecție este un obiect care include obiecte similare (dar nu neapărat), astfel încât se poate opera cu ansamblul lor. Acest lucru nu înseamnă că metodele sau proprietățile obiectelor (dacă sunt toate de acelaşi tip) se aplică tuturor elementelor colecției. Ca obiect separat, o colecție are proprietăți şi metode specifice (numărul de elemente, adăugarea unui nou element etc.).

De regulă, colecțiile definite în Office (există posibilitatea de a defini noi colecții) se remarcă prin aceea că au forma de plural a denumirii elementelor lor: **Workbooks** este colecția de obiecte **Workbook**, **Documents** este colecția de obiecte **Document** etc.

Elementele (membrii) colecției se pot identifica prin numărul de ordine (începând cu 1) sau prin nume (rezultă că ansamblul elementelor este ordonat). Astfel instrucțiunea

```
Presentations.Item("Perspective").Close
```

utilizată în PowerPoint produce activarea prezentării cu numele Perspective şi apoi o închide. Exemplul utilizează metoda Item pentru a returna elementul colecției de prezentări cu numele specificat. De regulă, această metodă este implicită, deci

```
Presentations("Perspective").Close
```

este o formă echivalentă.

Numărul de elemente ale colecției se pot afla prin proprietatea **Count**, se pot adăuga noi elemente prin metoda **Add** etc.

O utilizare frecventă a colecțiilor este parcurgerea tuturor elementelor într-o structură For Each…Next sau For…Next:

```
Public Sub DocScris()
For Each doc In Documents
               If doc.Words.Count > 1 Then
                              MsgBox doc.Name + Str(doc.Words.Count)
               End If
Next
End Sub
```

care, într-o aplicație Word, afişează numele tuturor documentelor deschise cu mai mult de un cuvânt scris.

Automatizarea acțiunilor prin folosirea obiectelor

Prin automatizarea unei acțiuni se înțelege scrierea unei proceduri care să producă, la executarea ei, acțiunea dorită. Execuția poate fi comandată direct sau ca răspuns la declaşarea unui eveniment.

Pentru a automatiza o acțiune în Microsoft Office, se va obține o referință la obiectul care dispune de conținutul şi funcționalitatea pe care le urmărim şi se vor aplica proprietățile şi metodele adecvate. Procesul poate necesita o succesiune de asemenea operații.

Obținerea unei referințe la un obiect

Pentru a obține o referință la un obiect trebuie să se construiască o expresie care ajunge să acceseze un obiect din modelul de obiecte şi apoi, utilizând proprietăți şi/sau metode, să se navigheze în sus sau în jos prin ierarhia de obiecte până când ajungem la obiectul dorit.

Proprietățile şi metodele utilizate pentru a returna punctul de start şi pentru a parcurge ierarhia de obiecte se numesc *accesori de obiecte* (object accessors) sau *accesori*.

Câteva idei utile pentru construirea expresiei care returnează referința la un obiect sunt următoarele:

- un loc obişnuit pentru a accesa modelul de obiecte este obiectul cu nivelul cel mai înalt, uzual obiectul **Application**. Se va utiliza proprietatea **Application** pentru a returna o referință la obiectul **Application**. Următoarea expresie returnează o referință la obiectul **Application** (pentru orice bibliotecă de obiecte care conține un obiect **Application**).

```
Application
```

- pentru a ajunge din vârful ierarhiei până la un obiect, se vor parcurge obiectele de pe toate nivelele, utilizând accesori care returnează un obiect din altul. De exemplu, proprietatea **Documents** a obiectului Word **Application** returnează obiectul colecție **Documents**, care reprezintă toate documentele deschise. Prin urmare următoarea expresie întoarce o referință la obiectul colecție **Documents**:

```
Application.Documents
```

- Există accesori direcți (shortcut accessors) care dau acces direct la obiecte din model fără să fie necesar un acces prin vârful ierarhiei. Asemenea accesori sunt **Documents**, **Workbooks**, **Presentations** care dau acces imediat la colecția de documente din Word, Excel şi PowerPoint. Există şi alte proprietăți cu rol de accesori direcți: **ActiveWindow**, **ActiveDocument**, **ActiveWorksheet**, **ActiveCell**. De exemplu, următoarea instrucțiune închide documentul Word activ:

```
ActiveDocument.Close
```

Observație. Se poate utiliza drept shortcut orice accesor care apare în zona **Members of** din Object Browser atunci când este selectat **<globals>** în zona **Classes**; adică nu trebuie să se returneze obiectul căruia i se aplică proprietatea sau metoda înaintea utilizării proprietății sau metodei, întrucât Visual Basic poate să determine din contextul în care se execută codul cărui obiect i se aplică proprietatea sau metoda respectivă.

- Pentru a returna un singur element al unei colecții se va utiliza proprietatea sau metoda **Item** cu numele sau numărul de ordine al elementului. Pentru cele mai multe colecții, **Item** este implicit, deci poate lipsi

```
Workbooks.Item("Vanzari")
Workbooks("Vanzari")
```

- Pentru a "urca" în ierarhia de obiecte, se utilizează, de obicei, proprietatea Parent a obiectului curent. De notat că proprietatea **Parent** poate returna uneori, în special dacă obiectul este membru al unei colecții, "bunicul" obiectului în locul părintelui (adică părintele colecției în locul colecției). De exemplu

```
Document.Parent
```

- returnează obiectul **Application** şi nu **Documents**.
- Prin funcția **TypeName** (executată eventual în Immediate Window) se poate găsi ce tip de obiect întoarce proprietatea **Parent** (funcția nu este limitată la această proprietate, vezi VB Help).

Aplicarea proprietăților şi metodelor

După obținerea unei referințe la obiectul urmărit, acestuia i se pot aplica proprietăți şi metode pentru a modifica valoarea unui atribut sau pentru a-l procesa. Se utilizează operatorul punct (".") pentru a separa expresia care returnează o referință la obiect de proprietatea sau metoda care se aplică obiectului. De exemplu

```
ActiveWindow.Left = 200
```

fixează poziția din stânga a ferestrei active utilizând proprietatea **Left** a obiectului **Window**, referința la acest obiect fiind returnată de accesorul direct **ActiveWindow**.

```
ActiveDocument.Close
```

închide documentul activ (în Word) utilizând metoda **Close** a obiectului **Document** la care returnează o referință accesorul **ActiveDocument**.

Proprietățile şi metodele pot avea argumente care să precizeze valorile sau acțiunile. Următorul exemplu Word utilizează metoda PrintOut cu specificarea paginilor care se tipăresc:

```
ActiveDocument.PrintOut From:=" 3", To:=" 7"
```

Este uneori necesar să se navigheze prin mai multe nivele în modelul de obiecte pentru a ajunge la ceea ce se consideră date reale în aplicație, cum ar fi valorile din celulele foii de calcul sau textul dintr-un document Word. Următoarele exemple Word arată cum se poate ajunge la text din vârful ierarhiei de obiecte:

- Proprietatea **Application** returnează o referință la obiectul **Application**.
- Proprietatea **Documents** a obiectului **Application** returnează o referință la colecția **Documents**.
- Metoda **Item** a colecției **Documents** returnează o referință la un songur obiect **Document**.
- Proprietatea **Words** a obiectului **Document** returnează o referință la colecția **Words**.
- Metoda **Item** a colecției **Words** returnează o referință la un singur obiect **Range**.
- Proprietatea **Text** a obiectului **Range** stabileşte textul itemului referit.

Astfel, următorul exemplu completează primul cuvânt din document

```
Application.Documents.Item(1).Words.Item(1).Text = "Primul "
```

Deoarece proprietatea **Documents** este o proprietate globală, poate fi utilizat fără calificativul **Application**; deoarece **Item** este proprietate sau metodă implicită pentru colecția de obiecte, nu trebuie enunțată explicit. Din aceste considerente, exemplul următor realizează exact aceeaşi acțiune ca şi exemplul precedent:

```
Documents(1).Words(1).Text = :Primul "
```

Pentru alte exemple de referințe şi de utilizare a metodelor şi proprietăților se vor urmări exemplificările de la capitolele următoare.

Ajutor în scrierea programelor

Pentru o imagine completă a uneltelor şi mecanismelor prin care mediul de programare VB susține activitatea de scriere a instrucțiunilor sursă se va citi şi capitolul dedicată mediului VBE.

Utilizarea Macro Recorder

Înregistrarea unui macro oferă un ajutor important atunci când se cunoaşte realizarea unei acțiuni în interfața utilizator a aplicației de bază şi se doreşte cunoaşterea obiectelor, proprietăților şi metodelor care pot să realizeze acea acțiune (sau ceva asemănător).

În general, codul generat de înregistrarea macro nu este foarte eficient şi robust, deoarece înregistrarea pleacă de la obiectul selectat în momentul startului şi realizează doar navigarea în restul modelului de obiecte. Orice utilizare ulterioară va necesita o selectare sau activare similară pentru a îndeplini acțiunea aşteptată. Trebuie să se considere codul înregistrat doar o primă schiță a procedurii, modificări ulterioare trebuind să producă o variantă mai clară şi mai robustă.

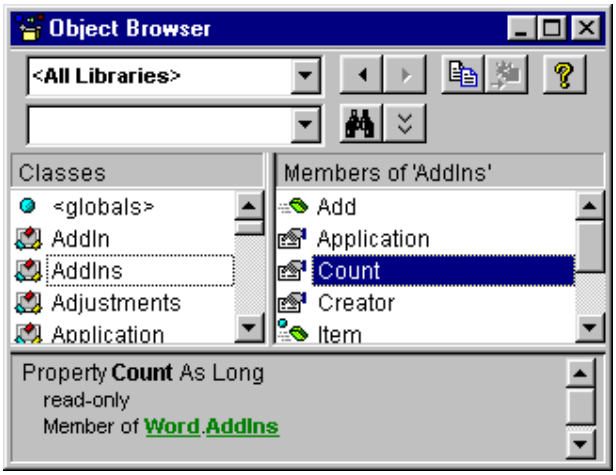
Codul generat este mai robust şi mai flexibil dacă va conține expresii care navighează prin ierarhia de obiecte fără să înceapă cu un obiect selectat sau activat. Idei în acest sens pot fi obținute din studierea exemplelor date în Visual Basic Help: poziționarea punctului de inserție pe o denumire de proprietate sau metodă şi acționarea tastei F1 afişează subiectul respectiv din Help.

O cale directă de accesare a fişierului de ajutor pentru un obiect este poziționarea în graficul care prezintă ierarhia de obiecte (specifică fiecărei aplicații) şi dublu click pe un obiect afişează subiectul dedicat obiectului în VB Help.

Exemplele prezentate în Help pot fi copiate, în mod uzual, utilizând Clipboard, în fereastra de cod.

Object Browser

Fiecare aplicație din Microsoft Office are o bibliotecă de obiecte (*object library* sau *type library*), care conține informații despre obiectele, proprietățile, metodele, evenimentele şi constantele predefinite ale aplicației. Pentru accesul la informația respectivă se poate utiliza Object Browser, unealtă din VBE.



Pentru a deschide Object Browser din VBE (în Excel, Word sau PowerPoint) sau dintr-un modul (Access), se alege Object Browser din meniul View.

În boxa Project/Library se alege numele bibliotecii care se consultă, sau <All libraries> pentru a vedea o listă completă. Dacă biblioteca dorită nu este în lista celor disponibile, se va crea o referință la această bibliotecă prin alegerile corespunzătoare în dialogul **References** (meniul **Tools**) al proiectului curent.

În boxa **Classes** se afişează numele tuturor obiectelor şi tipurilor enumerate (constantele predefinite) în bibliotecile referite.

Notă. O clasă este un tip, o descriere a unui obiect. Un obiect este o instanță efectivă a unei clase. Deseori aceşti termeni sunt utilizați unul în locul celuilalt, dacă nu se produc confuzii (uneori chiar şi atunci).

În boxa **Members of** se afişează toate proprietățile, metodele şi evenimentele proprii (asociate) clasei selectate în boxa **Classes**.

Selectarea unei intrări în listă poate fi completată cu F1 pentru a vedea textul ajutător, iar în zona inferioară (**Detail** pane) se afişează infromații privind sintaxa, starea read-only sau read-write, biblioteca unde aparține, tipul rezultatului returnat (dată sau obiect). Dacă o informație este de tip legătură, activarea acesteia produce informații suplimentare, lucru util pentru a deduce modul de navigare către obiect. În figura prezentată se vede proprietatea **Count** a clasei **AddIns**, proprietatea returnând o valoare de tip **Long**. Textul din zona Detail poate fi copiat (prin Clipboard) sau dus prin drag-and-drop într-o fereastră cod.

Legarea timpurie şi uneltele de construire a instrucțiunilor

Atunci când se creează într-o aplicație o variabilă obiect care se referă la un obiect furnizat de altă aplicație, Visual Basic trebuie să verifice că obiectul există şi că proprietățile şi metodele utilizate pentru obiect sunt specificate corect. Acest proces de verificare se numeşte **legare** (*binding*). Legarea poate să apară în timpul execuției proiectului (legare târzie) sau în timpul compilării (legare timpurie). Codul legat târziu este mai încet decât codul legat timpuriu. În plus, uneltele de ajutor în scrierea codului pot să lucreze corect doar în cazul legării timpurii.

Pentru a lega timpuriu codul se vor parcurge etapele:

Se stabileşte o referință la biblioteca de tipuri care conține obiectele referite (Tools - References).

- Se declară variabila obiect de un tip specific (de exemplu As Document şi nu As Object).
- Dacă se scrie cod care utilizează obiecte din mai multe biblioteci, se va specifica numele aplicației unde sunt declarate obiectele, mai ales dacă obiecte cu acelaşi nume există în mai multe biblioteci. (de exemplu As Excel.Window).

Dacă o proprietate sau o metodă utilizată returnează un tip generic Object şi nu un tip specific, atunci pentru legarea timpurie se va declara mai întâi o variabilă de tipul specific şi apoi se va atribui rezultatul generic returnat acestei variabile, după modelul

```
Dim testWs As Worksheet
Set testWs = Workbooks(1).Worksheets(1)
```

necesar deoarece metoda **Item** a obiectului **Worksheets** returnează un tip **Object** şi nu **Worksheet** (chiar dacă se referă la o foaie de calcul).

Programarea obiectelor altei aplicații

Se poate executa, într-o aplicație din Office, cod care să lucreze cu obiecte din altă aplicație. Pentru a realiza acest lucru, se va urmări schema următoare:

Se stabileşte o referinţă la biblioteca de tipuri a celeilate aplicaţii (meniul Tools - References).

Se declară variabile obiect care vor referi obiecte din altă aplicaţie cu tipuri specifice. Se va urmări calificarea fiecărui tip cu numele aplicaţiei care expune obiectul. Exemplul următor declară o variabilă care se referă la un document Word şi o variabilă care se referă la un caiet Exce:

```
Dim appWD As Word.Application, wbXL As Excel.Workbook
```

Se utilizează funcţia CreateObject cu identificatorul programatic OLE al obiectului cu care se doreşte să se lucreze în cealaltă aplicaţie, după modelul

```
Dim appWD As Word.Application
Set appWD = CreateObject("Word.Application.8")
appWD.Visible = True
```

Pentru informaţii asupra identificatorilor OLE se va vedea VB Help - "OLE Programmatic Identifiers".

Se aplică obiectului, conţinut în variabilă, proprietăţile şi metodele după modelul următor, care creează un nou document Word:

```
Dim appWD As Word.Application
Set appWD = CreateObject("Word.Application.8")
appWD.Documents.Add
```

La sfârşitul lucrului cu cealaltă aplicaţie, se va utiliza metoda Quit pentru a o închide, după modelul

```
appWD.Quit
```

Obiectele Microsoft Excel

Visual Basic suportă un set de obiecte care corespund direct elementelor din Microsoft Excel, cele mai multe identificabile după denumirea uzuală din mediul Excel. Astfel, obiectul Workbook reprezintă un caiet, obiectul Worksheet reprezintă o foaie de calcul iar obiectul Range reprezintă un domeniu de celule dintr-o foaie de calcul. Fiecare element din Microsoft Excel – caiet, foaie, diagramă, celulă etc. – poate fi reprezentat printr-un obiect în Visual Basic. Prin scrierea unor proceduri, care controlează aceste obiecte, se pot automatiza operaţiile efectuate în Excel.

Pentru a vedea modelul de obiecte pentru Microsoft Excel, se va căuta "Microsoft Excel Objects" în Help. Pentru a vedea fişierele de Help necesare se va urma calea: **Visual Basic Editor — Help — Contents and Index — (Contents** tab**) — Microsoft Excel Visual Basic Reference — Shortcut to Microsoft Excel Visual Basic Reference.** Fişierele sunt disponibile dacă la instalarea aplicaţiei s-a marcat boxa **Online Help for Visual Basic.**

Dintre cele peste 100 de obiecte care alcătuiesc ierarhia de obiecte Excel, vom prezenta în acest capitol doar pe cele mai importante. Prezentarea este simplificată şi din cauză că prezentarea obiectelor Word a conturat problematica modelelor de obiecte Office şi a fixat anumite reguli de operare cu aceste obiecte.

Obiectul Application

Cele mai multe proprietăţi ale obiectului **Application** Excel controlează atributele de vizualizare ale ferestrei aplicaţiei sau comporatrea globală a aplicaţiei. De exemplu, valoarea proprietăţii **DisplayFormulaBar** este True dacă bara de formule este vizibilă, iar valoarea proprietăţii **ScreenUpdating** este False dacă actualizarea ecranului este inhibată.

În plus, proprietăţile obiectului **Application** oferă acces la obiectele situate mai jos în ierarhie de obiecte (constituie ceea ce s-a numit *accesori*). Astfel, proprietatea **Windows** dă acces la colecţia **Windows** (reprezentând toate ferestrele deschise în aplicaţie), proprietatea **Workbooks** dă colecţia **Workbooks** a tutror caietelor deschise etc. Din această categorie enumerăm:

- **Charts**, colecţia tuturor foilor de tip chart,
- **Dialogs**, colecţia tuturor dialogurilor predefinite în mediul Excel,
- **Names**, colecţia tutror numelor create în caietul activ,
- **RecentFiles**, colecţia fişierelor utilizate recent (după lista din meniul File),
- **Sheets**, colecţia tutror foilor deschise în caietul activ,

- **Windows**,
- **Workbooks**,
- **Worksheets**, colecția tuturor foilor de calcul din caietul activ.

Returnarea unui obiect particular din colecție se efectuează după procedurile generale, explicate în capitolele introductive.

În categoria accesorilor mai pot fi încadrate proprietățile care returnează un obiect **Range**: **ActiveCell**, **Cells**, **Rows**, **Columns**, **Selection** (dacă este selectat un domeniu de celule).

Proprietățile **ActiveWorkbook, ActiveSheet, ActiveChart** şi **ActiveWindow** returnează obiectele care reprezintă elementele active corespunzătoare din Excel.

Anumite metode şi proprietăți care se aplică obiectului **Application** se aplică şi unor obiecte situate mai jos în ierarhie. Utilizarea acestor proprietăți şi metode la nivelul **Application** vor modifica toate caietele, foile deschise. De exemplu, metoda **Calculate** aplicată la nivelul **Application** produce recalcularea tuturor foilor, din toate caietele, pe când utilizată la nivel de **Workbook** sau de **Worksheet** produce recalcularea doar a foilor locale.

Obiectul Workbook

După cum se ştie, similarul unui document din Word este în Excel caietul (workbook). Deschiderea sau închiderea unui fişier în Excel implică deci deschiderea sau închiderea unui caiet. In Visual Basic, metodele utilizate la lucrul cu fişiere sunt metode ale obiectului **Workbook** sau ale colecției **Workbooks**.

Deschiderea unui Workbook

Pentru a deschide un caiet se utilizează metoda **Open**. Metoda este aplicată întotdeauna colecției **Workbooks**, returnată prin proprietatea globală cu aceeaşi denumire. Exemplul următor deschide caietul "Book1.xls" din folderul curent şi afişează apoi valoarea din prima celulă a primei foi:

```
Sub OpenBook1()
    Set myBook = Workbooks.Open(Filename:="Book1.xls")
    MsgBox myBook.Worksheets(1).Range("A1").Value
End Sub
```

Este de remarcat că obiectul **Workbook** returnat de metodă se referă la caietul deschis, care rămâne activ.

Asupra utilizării utilizării sau nu a căii pe care se găseşte fişierul se vor reciti cele spuse la deschiderea documentelor Word.

Există două foldere remarcabile pentru care se poate obține în mod automat calea: folderul cu fişierele Excel executabile şi folderul Library (creat automat la instalarea aplicației). Obținerea acestor căi se realizează prin proprietățile **Path** şi **LibraryPath** ale obiectului **Application**). Astfel

```
EXEPath = Application.Path & Application.PathSeparator
LibPath = Application.LibraryPath & Application.PathSeparator
```

returnează, respectiv, calea către fişierele executabile Excel şi calea către fişierele de bibliotecă. O cale returnată se termină cu separatorul adecvat sistemului pe care se execută aplicația, astfel încât codul este independent de platformă Windows sau Macintosh). Instrucțiunile

```
fName = LibPath & "Book1.xls"
Set myBook = Workbooks.Open(Filename:=fName)
```

considerate împreună cu atribuirea variabilei LibPath de mai sus, realizează deschiderea fişierului Book1.xls din folderul Library.

Se poate lăsa utilizatorului opțiunea de a decide aupra numelui fişierului care se deschide. Acest lucru se poate realiza prin metoda **GetOpenFilename** a obiectului **Application**. Metoda afişează cutia de dialog standard **Open,** dar, în loc să deschidă fişierul selectat, returnează un şir cu numele complet calificat al fişierului. Următorul exemplu demonstrează metoda:

```
Sub DemoGEtOpenFilename()
    Do
        fName = Application.GetOpenFilename
    Loop Until fName <> False
    MsgBox "Opening " & fName
    Set myBook = Workbook.Open (Filename:=fName)
End Sub
```

Metoda GetOpenFilename

Afişează dialogul **Open** şi returnează numele de fişier selectat fără a deschide efectiv fişierul.

expression.**GetOpenFilename**(FileFilter, FilterIndex, Title, ButtonText, MultiSelect)

unde

expression este o expresie care returnează un obiect Application.

FileFilter este de tip Variant, opţional. Este un şir specificând criteriile de filtrare a fişierelor listate în dialog. Şirul constă în perechi formate din şirul de filtrare şi din specificarea filtrului în format MS-DOS, toate elementele fiind separate prin virgule. În partea rezervată, două filtre MS-DOS sunt separate prin ";". Exemple: "Text Files (*.txt),*.txt,Add-In Files (*.xla),*.xla", "Visual Basic Files (*.bas; *.txt),*.bas;*.txt", implicit se consideră "All Files (*.*),*.*".

FilterIndex este de tip Variant, opţional. Specifică indexul criteriului de filtrare implicit. de la 1 la numărul de filtre specificat în *FileFilter*. Implicit se consideră 1.

Title este de tip Variant, opţional. Specifică titlul boxei de dialog. Implicit este "Open".

ButtonText este specific pentru Macintosh.

MultiSelect este de tip Variant, opţional. Este True atunci când se pot selecta mai multe nume de fişiere, False dacă este permisă selectarea unui singur fişier. Implicit este False. În cazul selecţiei multiple se va returna un tablou de denumiri (chiar dacă este selectat un singur fişier).

Metoda returnează numele fişierului selectat sau numele introdus de utilizator. În cazul când utilizatorul anulează boxa (prin **Cancel**), se returnează False. Metoda poate schimba atât folderul curent cât şi unitatea.

Crearea şi salvarea unui Workbook

Se creează un nou caiet prin aplicarea metodei **Add** la colecţia **Workbooks**. Valoarea returnată se va atribui (prin Set) unei variabile obiect pentru a putea referi noul caiet în program. Noul workbook devine activ.

Metoda Add (colecţia Workbooks)

Returnează un obiect **Workbook**. Sintaxa

expression.Add(Template)

unde

expression este o expresie care returnează un obiect **Workbooks**. (Metoda se poate aplica, cu parametri specifici, tuturor colecţiilor.)

Template este de tip Variant, opţional. Determină modul de creare a noului caiet. Dacă argumentul este un şir cu numele (posibil cu cale) unui fişier Excel, noul caiet este deschis după modelul fişierului specificat. Argumentul poate fi o constantă (de tipul enumerat XlWBATemplate), caz în care se va crea un caiet cu o singură foaie de tipul determinat de constantă. Valorile posibile sunt: xlWBATChart, xlWBATExcel4IntlMacroSheet, xlWBATExcel4MacroSheet sau xlWBATWorksheet. Dacă argumentul este omis, atunci se creează un caiet cu un număr de foi egal cu proprietatea **SheetsInNewWorkbook** a obiectului **Application**).

Salvarea unui caiet se efectuează prin metoda **SaveAs** (la prima salvare) sau prin metoda **Save**. Există, similar metodei **GetOpenFilename**, metoda **GetSaveAsFilename** (pentru **Application**).

Metoda SaveAs

are sintaxa

expression.**SaveAs**(Filename, FileFormat, Password, WriteResPassword, ReadOnlyRecommended, CreateBackup, AccessMode, ConflictResolution, AddToMru, TextCodePage, TextVisualLayout)

unde

expression returnează un obiect **Workbook**.

Filename, opţional, Variant. Conţine numele noului fişier, poate include o cale.

FileFormat, opţional, Variant. Specifică formatul de fişier utilizat la salvare. Lista formatelor admise (cele care se pot selecta şi la salvarea din Excel) se găseşte în Help la proprietatea **FileFormat**.

Password, opţional, Variant. Un şir unde capitalizarea este considerată (cel mult 15 caractere) care conţine parola de protejare a fisşierului.

WriteResPassword, opţional, Variant. Un şir care conţine parola necesară pentru scrierea fişierului. Dacă la deschidere nu se dă parola exactă, fişierul este deschis doar în citire.

ReadOnlyRecommended, opţional, Variant. Este True pentru a afişa, la deschidere, un mesaj cu recomandarea de a deschide fişierul doar în citire.

CreateBackup, opţional, Variant. Este True dacă se creează o copie backup.

AccessMode, opţional, Variant. Conţine modul de acces la workbook. Poate fi una dintre constantele (din tipul XlSaveAsAccessMode): xlShared (shared list), xlExclusive (exclusive mode) sau xlNoChange (nu se modifică modul de acces). Ultima valoare este cea implicită. Argumentul este ignorat dacă se salvează xlShared fără a schimba numele fişierului. Pentru schimbarea modului de acces se utilizează metoda **ExclusiveAccess**.

ConflictResolution, opţional, Variant. Specifică modul de rezolvare a conflictelor de schimbare în cazul când fişierul este shared. Poate fi una dintre constantele (de tip XlSaveConflictResolution): *xlUserResolution* (afişează un dialog privind conflictul şi rezolvarea)), xlLocalSessionChanges (acceptă automat modificările locale) sau xlOtherSessionChanges (acceptă celelalte schimbări în locul modificărilor locale). Prima constantă este valoarea implicită.

AddToMru, opţional, Variant. Este True dacă se adaugă numele fişierului la lista fişierelor utilizate recent. Implicit este False.

TextCodePage, *TextVisualLayout*, opţionale, Variant. Neutilizate în versiunea U.S. English.

Metoda Save

Salvează modificările caietului specificat.

expression.**Save**

unde

expression returnează un obiect Workbook.

Pentru marcarea unui fişier drept salvat fără a-l scrie efectiv pe disc, se va atribui valoarea True proprietăţii **Saved**.

Metoda GetSaveAsFilename

Similar metodei **GetOpenFilename**, această metodă afişează dialogul standard **Save As**, returnează un nume de fişier, dar nu salvează nici un fişier.

expression.**GetSaveAsFilename**(InitialFilename, FileFilter, FilterIndex, Title, ButtonText)

unde

expression este o expresie care returnează un obiect **Application**.

InitialFilename, opţional, Variant. Specifică numele de fişier propus. Dacă acest nume este omis, atunci se va utiliza numele caietului activ.

FileFilter, opţional, Variant. Şirul care specifică criteriul de filtrare. Pentru structura şirului se va revedea metoda **GetOpenFilename** de la deschiderea documentelor.

FilterIndex, opţional, Variant. Este indicele criteriului de filtrare, de la 1 la numărul de filtre dat la FileFilter. Implicit este 1.

Title, opţional, Variant. Titlul boxei de dialog.

ButtonText este specific Macintosh.

Metoda returnează numele de fişier selectat sau cel introdus de utilizator. Numele returnat poate include şi calea. Metoda returnează False dacă dialogul este închis de utilizator prin Cancel. Metoda poate schimba folderul sau unitatea curentă.

Următorul exemplu crează un nou caiet şi-l salvează prin metoda **GetSaveAsFilename**:

```
Sub CreateAndSave()
    Set newBook = Workbooks.Add
    Do
        fName = Application.GetSaveAsFilename
    Loop Until fName <> False
    newBook.SaveAs Filename:=fName
End Sub
```

Închiderea unui Workbook

Pentru a închide un workbook, se va aplica metoda **Close** a obiectului **Workbook**. Închiderea poate avea loc cu sau fără salvarea modificărilor.

Metoda Close

Produce închiderea obiectului. Aplicată colecţiei Workbooks are sintaxa

expression.**Close**

unde

expression returnează un obiect **Workbooks**. Dacă există modificări ale caietelor, se va afişa dialogul de interogare asupra eventualei salvări.

Aplicată obiectelor **Window** şi **Workbook** metoda are sintaxa

expression.**Close**(SaveChanges, FileName, RouteWorkbook)

unde

expression este o expresie care returnează un obiect **Workbook** sau **Window**.

SaveChanges este opțional, Variant. Dacă nu există modificări, argumentul este ignorat. Dacă există modificări în caiet dar caietul mai apare şi în altă fereastră deschisă, atunci argumentul este de asemenea ignorat. Dacă există modificări şi caietul nu mai apare în altă fereastră, atunci salvarea se efectuează după valorile: True – salvarea modificărilor sub numele dat la FileName sau dialog Save As; False – nu se salvează modificările; argument omis – interogare utilizator.

FileName este opțional, Variant. Salvează modificările sub acest nume.

RouteWorkbook este opțional, Variant. Dacă nu este indicată nici o rutare (nu există nici un **RoutingSlip** ataşat), argumentul este ignorat. Altfel, Excel efectuează rutarea documentului după valorile acestui argument: True – trimite caietul la următorul recipient; False – caietul nu este transmis mai departe; omis – interogarea utilizatorului asupra trimiterii.

Închiderea unui workbook din Visual Basic nu execută macrourile **Auto_Close** din workbook. Se va utiliza metoda **RunAutoMacros** pentru executarea macrourilor automate de închidere. Aceste macrouri sunt menținute în Excel din motive de compatibilitate, deci se referă la foi automatizate în versiuni Excel mai vechi.

Exemplul următor arată deschiderea unui caiet, modificări temporare ale caietului şi închiderea fără salvarea modificărilor:

```
Sub OpenChangeClose()
    Do
        fName = Application.GetOpenFilename
    Loop Until fName <> False
    Set myBook = Workbooks.Open (Filename:=fName)
    ' Aici se modifică foile de calcul
    myBook.Close SaveChanges:=False
End Sub
```

Obiectul Range

Prin intermediul unui obiect **Range** se poate referi o singură celulă, un domeniu de celule, o întreagă linie sau coloană, o selecție cu arii multiple sau un domeniu 3-D. Din acest motiv obiectul **Range** este oarecum neuzual prin aceea că poate reprezenta atât o singură celulă cât şi o mulțime de celule. Nu există un obiect colecție pentru **Range**, asă că un obiect **Range** poate fi gândit fie ca un obiect, fie ca o colecție, după situație.

Există foarte multe proprietăți şi metode care returnează un obiect Range:

ActiveCell	DirectDependents	RowFields
BottomRightCell	DirectPrecedents	RowRange
Cells	EntireColumn	Rows
ChangingCells	EntireRow	Selection
CircularReference	Next	TableRange1
Columns	Offset	TableRange2
CurrentArray	PageRange	TopLeftCell
CurrentRegion	Precedents	UsedRange

Dependents Range VisibleRange

Pentru specificarea exactă a acestor proprietăți şi metode se vor căuta subiectele respective în Help.

În continuare sunt menționate, mai mult prin exemple, moduri de lucru cu obiecte **Range**.

Referințe de tip A1 sau nume de domeniu

Unul dintre modurile uzuale de returnare a unui obiect Range este acela al utilizării unei referințe de tip A1 sau al unui nume definit.

inserarea unei valori într-o celulă:

```
Worksheets("Sheet1").Range("A1").Value = 3
```

inserarea unei formule într-o celulă:

```
Range("B1").Formula = "=5-10*RAND()"
```

inserarea aceleeaşi valori într-un întreg domeniu de celule:

```
Range("C1:E3").Value = 6
```

ştergerea conținutului unor celule:

```
Range("A1","E3").ClearContents
```

Stabilirea stilului bold pentru un domeniu numit (la nivel de workbook):

```
Range("myRange").Font.Bold = True
```

Atribuirea aceleeaşi valori fiecărei celule dintr-un domeniu numit (la nivel de foaie):

```
Range("Sheet1!yourRange").Value = 3
```

Setarea unei variabile obiect la un domeniu:

```
Set objRange = Range("myRange")
```

Este de menționat că expresiile care nu sunt calificate se referă la foaia curentă, deci multe din exemplele de mai sus nu ar opera dacă foaia curentă este o foaie de tip chart.

O cauză frecventă de erori este utilizarea proprietății **Range** ca argument al altei metode fără calificarea completă a obiectului **Worksheet** căruia i se aplică **Range**. Exemplul următor

```
Sub SortRange()
    Worksheets("Sheet1").Range("A1:B10").Sort _
            Key1:=Range("A1"), Order1:=xlDescending
End Sub
```

nu va funcționa corect decât dacă Sheet1 este foaia activă, altminteri calificarea argumentului Key1 nu este completă. Pentru o execuție independentă de context ar trebui folosit

```
Key1:=Worksheets("Sheet1").Range("A1")
```

Utilizarea indicilor de linii şi coloane

O celulă specifică poate fi returnată utilizând indicii numerici de linie şi coloană pentru celula referită.

Pentru a da o valoare celulei A1 se poate utiliza:

```
Worksheets("Sheet1").Cells(1,1).Value = 3
```

Pentru a insera o formulă în celula B1 din foaia activă:

```
Cells(1,2).Formula = "=5-10*RAND()"
```

Pentru a fixa o variabilă obiect la domeniul format din celula A1

```
Set objRange = Worksheets("Sheet1").Cells(1,1)
```

Referințele prin indici sunt utile mai ales la parcurgerea unui bloc de celule prin instrucțiuni de ciclare. Exemplul următor anulează toate celulele din domeniul A1:D10, cu o valoare mai mică decât 0.01:

```
Sub RoundToZero()
    For rwIndex = 1 to 10
        For colIndex = 1 to 4
            If Worksheets("Sheet1").Cells(rwIndex,colIndex) < 0.01 Then
                Worksheets("Sheet1").Cells(rwIndex,colIndex).Value = 0
            End If
        Next colIndez
    Next rwIndex
End Sub
```

În exemplul următor se arată o soluție la listarea, într-o foaie separată, a tuturor denumirilor create în caietul activ şi a domeniilor referite de acestea.

```
Sub ListNames()
    Set newSheet = Worksheets.Add
    I = 1
    For Each nm in ActiveWorkbook.Names
        NewSheet.Cells(i,1).Value = nm.Name
        NewSheet.Cells(i,2).Value = "' " & nm.RefersTo
    Next nm
    NewSheet.Columns("A:B").AutoFit
End Sub
```

Utilizarea proprietății Offset

Atunci când este necesară referirea la un domeniu prin deplasări relative la alt domeniu de celule, se poate utiliza proprietatea **Offset**, a obiectului **Range**, care în argumentele *RowOffset* şi *ColumnOffset* arată deplasarea față de obiectul **Range** curent. Este returnat un nou obiect **Range**.

Exemplul următor determină câteva tipuri de date din celulele domeniului A1:A10, tipurile determinate fiind înscrise, ca text, în celula corespunzătoare din dreapta, B1:B10.

```
Sub ScanColumn ()
    For Each c In Worksheets("Sheet1").Range("A1:A10").Cells
        If Application.IsText(c.Value) Then
            c.Offset(0,1).Value = "Text"
        ElseIf Application.IsNumber (c.Value) Then
            c.Offset(0,1).Value = "Number"
        ElseIf Application.IsLogical (c.Value) Then
            c.Offset(0,1).Value = "Boolean"
        ElseIf Application.IsError (c.Value) Then
            c.Offset(0,1).Value = "Error"
        ElseIf c.Value = "" Then
            c.Offset(0,1).Value = "(blank cell)"
        End If
    Next c
End Sub
```

Utilizarea proprietăților CurrentRegion şi UsedRange

Aceste două proprietăți, explicate în continuare, sunt utile atunci când nu se ştie de la început cât de mare este domeniul pe care se operează.

Prin **regiunea curentă** se înțelege un domeniu dreptunghiular de celule, limitat de linii şi coloane goale, eventual de marginile foii de calcul şi de linii şi coloane goale. Proprietatea **CurrentRegion** se aplică unui obiect **Range** şi pot fi mai multe regiuni curente pe o foaie de calcul, după obiectul **Range** căruia i se aplică proprietatea. Proprietatea returnează un obiect **Range**, reflectând extensia, în sensul prezentat mai sus, al obiectului **Range** căruia i se aplică proprietatea.

Domeniul utilizat este determinat de celule nevide situate cel mai la stânga sus şi cel mai la dreapta jos într-o foaie de calcul. Un asemenea domeniu conține toate celule nevide din foaie, ca şi celule vide interpuse până la completarea unui domeniu dreptunghiular şi este unic pe o foaie de calcul. Este natural ca proprietatea **UsedRange** să se aplice obiectului **Worksheet** şi nu unui obiect **Range**. Proprietatea returnează un obiect **Range**.

Următorul exemplu aplică celulelor cu valori numerice dintr-o listă, care începe în celula A1, formatul numeric 0.0:

```
Sub FormatRange ()
    Set myRange = Worksheets("Sheet1").Range("A1").CurrentRegion
    MyRange.NumberFormat = "0.0"
End Sub
```

Exemplul care urmează presupune că foaia activă conține date dintr-un experiment desfăşurat în timp: prima coloană conține datele calendaristice, a doua coloană conține ora înregistrării valorilor, coloanele a treia şi a patra conțin măsurătorile experimentului. Procedura prezentată combină primele două coloane într-o singură valoare de tip Date, converteşte valoarea obținută din GMT (Greenwich Mean Time) în PST (Pacific Standard Time) şi le formatează. Deoarece nu se ştie dacă există şi coloane goale între cele patru coloane cu date, se utilizează **UsedRange**.

```
Sub ConvertDates ()
    Set myRange = ActiveSheet.UsedRange
    myRange.Columns("C").Insert
    Set dateCol = myRange.Columns("C")
    For Each c In dateCol.Cells
        If c.Offset(0,-1).Value <>"" Then
            c.FormulaR1C1 = "=RC[-2]+RC[-1] - (8/24)"
        End If
    Next c
    dateCol.NumberFormat = "mmm-dd-yyyy hh:mm"
    dateCol.Copy
    dateCol.PasteSpecial Paste:=xlValues
    myRange.Columns("A:B").Delete
    dateCol.AutoFit
```

```
End Sub
```

Există şi alte proprietăţi şi metode care produc fie subdomenii, fie supradomenii pornind de la un obiect **Range**. Printre acestea enumerăm: **Areas, Cells, Columns, EntireColumn, EntireRow, Range** şi **Rows**.

Parcurgerea unui domeniu de celule

Dintre multiplele moduri de parcurgere a celulelor dintr-un domeniu, se prezintă parcurgerile prin instrucţiunile **For Each … Next** şi **Do … Loop**, unele fiind deja utilizate în exemplele anterioare.

Utilizarea instrucţiunii For Each … Next

Acesta este modul recomandat de parcurgere a elementelor unei colecţii.

Un exemplu anterior devine

```
Sub RoundToZero ()
    For Each r In Worksheets(Sheets1").Range("A1:D10").Cells
        If Abs(r.Value) < 0.01 Then
            r.Value = 0
        End If
    Next r
End Sub
```

Pentru ca operaţiunea anterioară să aibă loc pe un domeniu selectat de utilizator, se poate utiliza metoda **InputBox**, specificându-i utilizatorului să selecteze un domeniu de celule. Metoda returnează un obiect **Range** care reprezintă selecţia. Codul este completat cu instrucţiuni de tratare a erorilor uzuale.

```
Sub RoundToZero ()
    Worksheets("Sheet1").Activate
    On Error GoTo PressedCancel
    Set r = Application.InputBox( _
             Prompt:="Select a range of cells", _
             Type:=8)
    On Error GoTo 0
    For Each c In r.Cells
        If Abs(c.Value) < 0.01 Then
            c.Value = 0
        End If
    Next c
    Exit Sub

PressedCancel:
    Resume
End Sub
```

Dacă nu se doreşte selectarea de către utilizator a domeniului procesat, se poate utiliza proprietatea **CurrentRegion** sau proprietatea **UsedRegion** pentru a returna obiectul **Range** prelucrat. De exemplu, dacă se ştie că domeniul începe cu celula A1 şi nu include linii sau coloane vide, atunci se poate utiliza

```
Set r = Worksheets("Sheet1").Range("A1").CurrentRegion
```

pentru a returna întregul domeniu (compact) de celule care se prelucrează.

Următoarele două exemple arată cum se poate ascunde fiecare a doua coloană din domeniul utilizat în Sheet1. Primul exemplu, utilizând For Each…Next

```
Sub HideColumns ()
    Set r = Worksheets("Sheet1").UsedRange
    For Each col In r.Columns
        If col.Column Mod 2 = 0 Then
            col.Hidden = True
        End If
    Next col
End Sub
```

Al doilea exemplu, utilizând For…Next:

```
Sub HideColumns ()
    Set r = Worksheets("Sheet1").UsedRange
    For i = 1 To r.Columns.Count
        If i Mod 2 = 0 Then
            r.Columns(i).Hidden = True
        End If
    Next i
End Sub
```

Utilizarea instrucţiunii Do…Loop

Atunci când procesarea unui domeniu modifică domeniul (de exemplu prin ştergerea unor linii/coloane), utilizarea instrucţiunii For Each…Next nu produce cele mai bune rezultate. Soluţia este atunci utilizarea instrucţiunii Do…Loop. Exemplul următor sortează o listă şi elimină liniile elementelor duplicate:

```
Sub RemoveDuplicates ()
    Worksheets("Sheet1").Range("A1").Sort _
                Key1:=Worksheets("Sheet1").Range("A1")
```

```
    Set currentCell = Worksheets("Sheet1").Range("A1")
    Do While Not IsEmpty(currentCell)
        Set nextCell = currentCell.Offset(1,0)
        If nextCell.Value = currentCell.Value Then
            currentCell.EntireRow.Delete
        End If
        Set currentCell = nextCell
    Loop
End Sub
```

Este de notat că structura repetitivă poate fi înlocuită prin

```
Do While currentCell.Value <> ""
    ' instrucţiunile de eliminare a liniilor cheilor duplicate
Loop
```

Utilizarea proprietății Address

Aplicarea proprietății Address returnează adresa de celule a domeniului, adresa fiind sub forma de şir de caractere. Această utilizare este utilă, în general, pentru verificare şi depanarea codului. Exemplul următor arată o formă de completare a unei proceduri anterioare cu instrucțiuni de control a mersului programului

```
Sub HideColumns ()
    Set r = Worksheets("Sheet1").UsedRange
    MsgBox r.Address ' doar pentru depanare
    For i = 1 To r.Columns.Count
        If i Mod 2 = 0 Then
            r.Columns(i).Hidden = True
            MsgBox r.Columns(i).Address ' doar pentru depanare
        End If
    Next i
End Sub
```

Acelaşi efect se poate obține prin stabilirea unor expresii de urmărire (watch expressions) de forma *r.Address* şi *r.Columns(i).Address,* valorile respective pot fi examinate în fereastra **Immediate**. Pentru o discuție mai pe larg se va studia capitolul dedicat depanării şi manevrării erorilor.

Evenimentele din Excel

O bună parte din codul scris într-o aplicație este conținut în proceduri de răspuns la evenimente. Cunoaşterea evenimentelor şi alegerea unor răspunsuri adecvate produc o aplicație senzitivă, vie, care interacționează bine cu utilizatorul.

În Microsoft Excel se pot scrie proceduri eveniment la nivelurile: worksheet, chart, workbook şi application. În plus față de versiuni anterioare, sunt posibile şi proceduri eveniment cu argumente.

Procedurile de răspuns la evenimente la nivelurile Worksheet şi Workbook sunt create în mod implicit pentru orice foaie de calcul, foaie de diagramă sau caiet. Pentru a scrie proceduri de răspuns la evenimentele de la nivelul Chart sau pentru Application, trebuie să se creeze un nou obiect utilizând cuvântul cheie **WithEvents** într-un modul clasă. (vezi discuția din secțiunea dedicată subiectului în acest capitol).

Permiterea şi inhibarea evenimentelor

În mod uzual, toate evenimentele sunt permise. Cu alte cuvinte evenimentele au loc, sunt recunoscute ca atare şi se execută procedurile corespunzătoare fiecărui eveniment.

În cazul când nu se doreşte executarea procedurii de răspuns, acest lucru este controlat prin inhibarea evenimentului, cu efectul nerecunoaşterii evenimentului de către sistem şi, drept urmare, neexecutarea procedurii asociate.

Proprietatea EnableEvents, a obiectului Application, poate primi valoarea True sau False după cum evenimentele sunt permise sau inhibate.

Următorul exemplu execută salvarea caietului fără producerea evenimentului BeforeSave:

```
Application.EnableEvents = False
ActiveWorkbook.Save
Application.EnableEvents = True
```

Utilizarea evenimentelor

Completarea procedurilor implicite de răspuns la evenimente se efectuează prin accesul la codul procedurilor şi scrierea de cod în mod uzual.

Pentru a vedea procedurile de eveniment ale unei foi (de calcul sau diagramă):

- click dreapta pe cotorul foii (pe bara de jos, unde se văd cotoarele tuturor foilor din caietul activ), comanda **View Code** din meniul contextual, alegerea numelui evenimentului în lista derulantă **Procedure**, sau
- meniul **Tools**, comanda **Macro** şi selectarea opţiunii **Visual Basic Editor**. Se selectează foaia dorită în Project Explorer, butonul **View Code** şi se alege numele evenimentului din lista **Procedure**.

Evenimentele obiectului Worksheet

Eveniment	Descriere
Activate	Apare atunci când utilizatorul activează foaia. Acest eveniment se va utiliza în locul proprietăţii **OnSheetActivate**
BeforeDoubleClick	Apare atunci când utilizatorul execută un dublu click într-o celulă a foii. Se va utiliza în locul proprietăţii **OnDoubleClick**.
BeforeRightClick	Apare atunci când utilizatorul execută un click dreapta într-o celulă a foii.
Calculate	Apare când utilizatorul recalculează foaia. Acest eveniment se va utiliza în locul proprietăţii **OnCalculate**.
Change	Apare atunci când utilizatorul schimbă o formulă dintr-o celulă. Se va utiliza în locul proprietăţii **OnEntry**.
Deactivate	Apare atunci când foaia este activă şi utilizatorul activează o altă foaie. Nu apare atunci când utilizatorul mută focusul de la o fereastră la altă fereastră a aceleeaşi foi. Acest eveniment se va utiliza în locul proprietăţii **OnSheetDeactivate**.
SelectionChange	Apare atunci când utilizatorul selectează o celulă din foaie.

O prezentare completă şi exemple se găsesc în intrările respective din Help.

Exemplu

În codul care urmează, se reajustează dimensiunea coloanelor la fiecare recalculare:

```
Private Sub Worksheet_Calculate ()
    Columns("A:F").AutoFit
End Sub
```

Este de remarcat că modelul procedurii este accesat printr-una din tehnicile descrise la "Utilizarea evenimentelor".

Evenimentele obiectului Chart

Declanşate atunci când utilizatorul activează sau modifică o diagramă, evenimentele recunoscute de obiectul Chart sunt prezentate în tabelul următor.

Eveniment	Descriere

Activate	Apare atunci când utilizatorul activează foaia diagramă (nu apare la diagramele scufundate). Acest eveniment se va utiliza în locul proprietății **OnSheetActivate**
BeforeDoubleClick	Apare atunci când utilizatorul execută un dublu click pe diagramă. Se va utiliza în locul proprietății **OnDoubleClick**.
BeforeRightClick	Apare atunci când utilizatorul execută un click dreapta pe diagramă.
Calculate	Apare când utilizatorul reprezintă în diagramă date noi sau modificate.
Deactivate	Apare atunci când foaia este activă şi utilizatorul activează o altă foaie. Nu apare atunci când utilizatorul mută focusul de la o fereastră la altă fereastră a aceleeaşi foi. Acest eveniment se va utiliza în locul proprietății **OnSheetDeactivate**.
DragOver	Apare atunci când utilizatorul draghează date peste diagramă.
DragPlot	Apare atunci când utilizatorul draghează un domeniu de celule peste diagramă.
MouseDown	Apare atunci când utilizatorul execută un click cu un buton al mouse-ului în timp ce pointerul acestuia este poziționat pe diagramă.
MouseMove	Apare la mişcarea pointerului mouse-ului peste diagramă.
MouseUp	Apare atunci când utilizatorul eliberează un buton al mouse-ului în timp ce pointerul acestuia este poziționat pe diagramă.
Resize	Apare la redimensionarea diagramei.
Select	Apare la selectarea unui element al diagramei.
SeriesChanges	Apare atunci când utilizatorul modifică valoarea unei punct de pe diagramă.

Evenimentele foilor de diagrame sunt permise în mod implicit. Pentru a scrie proceduri de eveniment pentru diagramele scufundate, trebuie să se creeze un nou obiect utilizând **WithEvents** într-un modul de clasă.

Exemplu

Se schimbă culoarea chenarului unui punct atunci când utilizatorul schimbă valoarea punctului:

```
Private Sub Chart_SeriesChange (ByVal SeriesIndex As Long, _
                                ByVal PointIndex As Long)
    Set p = ActiveChart.SeriesCollection(SeriesIndex).Points(PointIndex)
    p.Border.ColorIndex = 3
```

```
End Sub
```

Evenimentele obiectului Workbook

Aceste evenimente se declanşează atunci când utilizatorul schimbă un caiet sau orice foaie din caietul respectiv.

Eveniment	Descriere
Activate	Apare atunci când utilizatorul activează caietul.
AddInInstall	Apare atunci când utilizatorul instalează caietul ca un add-in. Se va utiliza în locul macro-ului Auto_Add.
AddInUninstall	Apare atunci când utilizatorul dezinstalează caietul ca un add-in. Se va utiliza în locul macro-ului Auto_Remove.
BeforeClose	Apare înaintea închiderii caietului. Se va utiliza în locul macro-ului Auto_Close.
BeforePrint	Apare înaintea tipăririi caietului.
BeforeSave	Apare înainte ca utilizatorul să salveze foaia. Acest eveniment se va utiliza în locul proprietăţii **OnSave**.
Deactivate	Apare atunci când caietul este activ şi utilizatorul activează un alt caiet.
NewSheet	Apare după ce utilizatorul creează o nouă foaie.
Open	Apare la deschiderea caietului. Evenimentul se va utiliza în locul macroului Auto_Open.
SheetActivate	Apare la activarea unei foi din caiet. Se va utiliza în locul proprietăţii **OnSheetActivate**.
SheetBeforeDoubleClick	Apare la dublu click pe o celulă (nu este utilizat cu foile diagramă). Se va utiliza în locul proprietăţii **OnDoubleClick**.
SheetBeforeRightClick	Apare la click dreapta pe o celulă a unei foi din caiet (nu este utilizat cu foile diagramă).
SheetCalculate	Apare la recalcularea unei foi (nu este utilizată cu foile diagramă). Se utilizează în locul proprietăţii **OnCalculate**.
SheetChange	Apare la modificarea formulei dintr-o celulă (nu este utilizată cu foile diagramă). Se utilizează în locul proprietăţii **OnEntry**.

SheetDeactivate	Apare la activarea altei foi din caiet. Se utilizează în locul proprietății **OnSheetDeactivate**.
SheetSelectionChange	Apare la modificarea selecției dintr-o foaie de calcul (nu funcționează cu foile diagramă).
WindowActivate	Apare atunci când utilizatorul mută focusul pe orice fereastră a caietului. Se utilizează în locul proprietății **OnWindow**.
WindowDeactivate	Apare atunci când utilizatorul mută focusul în afara oricărei ferestre a caietului. Se utilizează în locul proprietății **OnWindow**.
WindowResize	Apare atunci când utilizatorul deschide, redimensionează, maximizează sau minimizează orice fereastră a caietului.

Pentru explicații se vor studia intrările corespunzătoare din Help.

Exemplu

Deschiderea caietului maximizează fereastra aplicației Excel:

```
Sub Workbook_Open ()
    Application.WindowState = xlMaximized
End Sub
```

Evenimentele obiectului Application

Aceste evenimente se declanşează la crearea/deschiderea unui caiet sau atunci când este modificată orice foaie din orice caiet deschis.

Eveniment (pentru Application)	**Descriere**
NewWorkbook	Apare la crearea unui nou caiet.
SheetActivate	Apare atunci când utilizatorul activează o foaie dintr-un caiet deschis. Se va utiliza în locul proprietății **OnSheetActivate**.
SheetBeforeDoubleClick	Apare la dublu click pe o celulă dintr-un caiet deschis (nu este utilizat cu foile diagramă). Se va utiliza în locul proprietății **OnDoubleClick**.
SheetBeforeRightClick	Apare la click dreapta pe o celulă a unei foi dintr-un caiet deschis (nu este utilizat cu foile diagramă).
SheetCalculate	Apare la recalcularea unei foi (nu este utilizată cu foile diagramă). Se utilizează în locul proprietății **OnCalculate**.
SheetChange	Apare la modificarea formulei dintr-o celulă (nu este utilizată cu foile diagramă). Se utilizează în locul proprietății **OnEntry**.

SheetDeactivate	Apare la activarea altei foi dintr-un caiet. Se utilizează în locul proprietății **OnSheetDeactivate**.
SheetSelectionChange	Apare la modificarea selecției dintr-o foaie de calcul (nu funcționează cu foile diagramă).
WindowActivate	Apare atunci când utilizatorul mută focusul pe orice fereastră deschisă în aplicație. Se utilizează în locul proprietății **OnWindow**.
WindowDeactivate	Apare atunci când utilizatorul mută focusul în afara oricărei fereastre a aplicației. Se utilizează în locul proprietății **OnWindow**.
WindowResize	Apare atunci când utilizatorul redimensionează, maximizează sau minimizează orice fereastră deschisă în aplicație.
WorkbookActivate	Apare atunci când se mută focusul pe un caiet deschis
WorkbookAddInInstall	Apare la instalarea unui workbook ca un add-in.
WorkbookAddInUninstall	Apare la dezinstalarea unui workbook ca un add-in.
WorkbookBeforeClose	Apare înainte ca un caiet deschis să fie închis.
WorkbookBeforePrint	Apare înainte ca un caiet deschis să fie tipărit.
WorkbookBeforeSave	Apare înainte ca un caiet deschis să fie salvat.
WorkbookDeactivate	Apare atunci când utilizatorul mută focusul în afara unui caiet deschis.
WorkbookNewSheet	Apare la adăugarea unei noi foi la un caiet deschis.
WorkbookOpen	Apare atunci când utilizatorul deschide un caiet.

Utilizarea modulelor clasă cu evenimente

Deoarece diagramele scufundate într-o foaie de calcul şi obiectul **Application** nu au evenimente permise în mod implicit, trebuie să se urmeze următoarele etape pentru a utiliza evenimentele recunoscute de aceste obiecte.

- Se creează un modul de tip clasă şi se declară un obiect de tip **Chart** sau **Application** cu evenimente. Pentru crearea modulului clasă se dă comanda **Class Module** din meniul **Insert**.
- Pentru permiterea evenimentelor obiectului **Application** se adaugă declarația

```
Public WithEvents App As Application
```

- Obiectul nou creat apare în boxa **Object** din modulul clasă şi se pot scrie procedurile evenimentelor pentru noul obiect.

- Se conectează obiectul declarat în modul la obiectul **Application**. Pentru această operațiune, în orice modul se dă instrucțiunea

```
Public X As New EventClass
```

unde EventClass este numele dat, de exemplu, modulului clasă creat, similar pentru X.

- după crearea instanței X a obiectului EventClass se poate stabili obiectul App al clasei EventClass egal cu obiectul **Application** Microsoft Excel.

```
Sub InitializeApp ()
    Set X.App = Application
End Sub
```

- După executarea procedurii de inițializare, obiectul App din modulul EventClass punctează către obiectul **Application** Microsoft Excel şi procedurile eveniment din modulul clasă vor fi executate la declanşarea evenimentelor.

Deşi procedura poate părea laborioasă, ideea poate fi utilizată pentru ca aceleaşi proceduri eveniment să fie asociate mai multor obiecte.

Să presupunem că am efectuat etapele precedente pentru un obiect diagramă. S-a utilizat astfel declarația

```
Public WithEvents cht As Chart
```

în etapa 2 şi codul următor

```
Dim C1 As New EventClass
Dim C2 As New EventClass
Sub InitializeCharts ()
    Set C1.cht = Worksheets (1).ChartObjects(1).Chart
    Set C2.cht = Worksheets (1).ChartObjects(2).Chart
End Sub
```

pentru inițializare.

Aceeaşi tehnică se poate utiliza şi pentru obiectele **Worksheet** şi **Workbook** pentru a utiliza evenimentele noii clase cu mai multe foi de calcul, în plus față de evenimentele implicite.

Tratarea erorilor şi depanarea programelor

Acest capitol arată cum să se utilizeze uneltele de depanare şi verificare a programelor oferite de Visual Basic Editor. Sunt prezentate şi modurile în care se pot controla erorile din execuție (*run-time errors*) apărute în urma încercării de a executa operațiuni interzise.

Informațiile din acest capitol se aplică proiectelor VBA din Microsoft Excel.

Cum se gestionează erorile

În mod ideal, procedurile Visual Basic n-ar trebui să conțină cod de tratare a erorilro. Realitatea arată însă că probleme hardware sau acțiuni neanticipate ale utilizatorului pot să producă erori în execuție, ca urmare programul se opreşte şi există puține şanse ca utilizatorul să continue execuția aplicației. Sunt posibile şi erori care deşi nu opresc execuția produc rezultate ulterioare eronate.

De exemplu, procedura următoare returnează **True** dacă fişierul specificat există şi **False** dacă nu, dar nu conține instrucțiuni de tratare a erorilor.

```
Function FileExists (filename) As Boolean
         FileExists = (Dir(filename) <>"")
End Function
```

Funcția **Dir** returnează primul fişier care se potriveşte numelui specificat sau un şir de lungime zero în cazul când nu există nici o potrivire.

Codul anterior pare a fi corect şi acoperitor pentru toate situațiile posibile. Totuşî, dacă litera de drive specificată nu este validă, apare eroarea "Device unavailable". Dacă unitatea este de floppy disk, funcția va funcționa corect doar dacă în unitate este o dischetă şi poarta este închisă. În caz contrar Visual Basic va semnala eroarea "Disk not ready" şi execuția se opreşte.

Pentru a ocoli asemenea situații, trebuie să se utilizeze modalitățile de error-handling din VBE pentru a intercepta erorile (sau "a prinde" - *trapping*) şi a executa acțiuni corective. La apariția unei erori, Visual Basic stabileşte diferite proprietăți ale obiectului reprezentând eroare, **Err**, cum ar fi numărul erorii, descrierea etc. Obiectul **Err** poate fi utilizat într-o rutină de tratare a erorii astfel încât aplicația poate răspunde inteligent la o situație de eroare.

De exemplu, probleme legate de unitate, cum ar fi o unitate invalidă sau o unitate de floppy fără dischetă, pot fi tratate după modelul următor.

```
Function FileExists (filename) As Boolean
         Dim Msg As String
         ' Trecerea la tratarea erorilor
         On Error GoTo CheckError
                  FileExists = (Dir(filename) <> "")
                  ' nu a apărut nici o eroare
                  Exit Function
CheckError:                          ' secvenţa de tratare a erorii apărute
         ' definirea constantelor care reprezintă eroarea
         Const mnErrDiskNotReady = 71, mnErrDeviceUnavailable = 68
         ' vbExclamation, vbOK, vbCancel, vbCritical şi vbOKCancel
         ' sunt constante definite în biblioteca VBA
         If (Err.Number = mnErrDiskNotReady) Then
                  Msg = "Put a floppy disk in the drive and close the door"
                  ' Afişarea mesajului
                  If MsgBox(Msg, vbExclamation & vbOKCancel) = vbOK Then
                           Resume
                  Else
                           Resume Next
                  End If
         ElseIf Err.Number = mnErrDeviceUnavailable Then
                  Msg = "This drive or path does not exist: " & filename
                  MsgBox Msg, vbExclamation
                  Resume Next
         Else
                  Msg = "Unexpected error #" & Str(Err.Number) & " occurred: " _
                           & Error.Description
                  ' Afişarea mesajului
                  MsgBox Msg, vbCritical
                  Stop
         End If
         Resume
End Function
```

În acest cod, proprietatea **Number** a obiectului **Err** conţine numărul asociat cu eroarea aparută; proprietatea **Description** conţine o scurtă descriere a erorii. Instrucţiunea **Resume** returnează controlul la instrucţiunea care a produs eroarea iar **Resume Next** returnează controlul la instrucţiunea următoare celei care a produs eroarea.

Proiectarea unei rutine de tratare a erorilor

O secvenţă de tratare a erorilor, *error handler*, este o rutină care captează şi răspunde la erori în aplicaţie. Asemenea secvenţe se pot adăuga la orice procedură unde se anticipează posibilitatea unei erori (se poate presupune că orice instrucţiune Visual Basic poate produce o eroare, cu excepţia cazurilor când se ştie explicit că acest fapt nu poate avea loc). Procesul de proiectare a unui error handler necesită trei paşi:

1. Stabilirea, sau permiterea, unei captări de erori prin specificarea locului din program unde se va efectua transferul controlului atunci când apare o eroare.

Instrucţiunea **On Error** permite captarea erorii şi transferul controlului execuţiei la eticheta specificată.

2. Scrierea unei rutine de tratare a erorilor, care să răspundă tuturor erorilor anticipate. Dacă execuţia şi-a transferat controlul în această secvenţă, se zice că secvenţa este activă (capcana de prindere a erorii este activă).

3. Ieşirea (părăsirea) secvenţei error-handling.

Detalii sunt oferite în secţiunile care urmează.

Stabilirea capcanei de erori

O capcană de erori este activată atunci când se execută o instrucţiune **On Error**, care specifică o secvenţă de tratare a erorilor. Capcana rămâne deschisă atât timp cât procedura care o conţine este activă, adică până când este executată o instrucţiune **Exit Sub, Exit Function, Exit Property, End Sub, End Function** sau **End Property** din procedura care include secvenţa de tratare a erorilor. Deşi numai o capcană de erori poate fi permisă la un moment dat în orice procedură, se pot crea mai multe capcane alternative care să fie activate la diferite momente. O capcană de erori poate fi inhibată prin utilizarea instrucţiunii **On Error GoTo 0**.

Pentru activarea unei capcane de erori se utilizează instrucţiunea **On Error GoTo** *line*, unde *line* este eticheta care identifică secvenţa de instrucţiuni, din procedură, care tratează erorile.

Scrierea unei rutine de tratare a erorilor

Primul pas în scrierea unei rutine error-handling este adăugarea unei etichete de linii pentru marcarea începutului secvenţei de tratare a erorilor. O convenţie comună este aceea de a plasa codul error-handler la sfârşitul procedurii şi a-l preceda de o instrucţiune **Exit Sub, Exit Function** sau **Exit Property**. Aceasta permite procedurii să nu execute codul de tratare a erorilor atunci când nu apar erori.

Corpul rutinei de tratare a erorilor conține cod de identificare a erorii apărute, prin structuri **Select Case** sau **If...Then...Else**. Este obligatoriu ca o alternativă să se refere la erorile neanticipate şi care nu sunt tratate individual.

Proprietatea **Number** a obiectului **Err** conține un cod numeric reprezentând cea mai recentă eroare. Utilizarea obiectului **Err** în combinație cu structurile **Select Case** sau **If...Then...Else**, se pot lua acțiuni specifice fiecărei erori.

Ieşirea din rutina de tratare a erorilor

Următorul tabel cuprinde instrucțiunile prin care se poate ieşi dintr-o secvență error-handling. Acest fapt înseamnă, în general, cedarea controlului execuției către altă instrucțiune din procedura activă.

Instrucțiunea	Descriere
Resume [0]	Execuția programului se reia cu instrucțiunea care a cauzat eroarea sau cel mai recent apel al procedurii conținând rutina error-handling. Este utilizată pentru a repeta o operațiune după ce s-a corectat situația care a produs eroarea.
Resume Next	Reia execuția programului cu instrucțiunea imediat următoare celei care a cauzat eroarea. Dacă eroarea a apărut în afara procedurii care conține codul de eroare, execuția se reia cu instrucțiunea imediat următoare apelului la procedura unde a apărut eroarea, dacă procedura apelată nu are activată o secvență de eroare.
Resume *line*	Reia execuția programului cu instrucțiunea având eticheta specificată în *line*, unde *line* este o etichetă de linie (sau un număr de linie diferit de zero) care se găseşte în aceeaşi procedură ca şi error-handler-ul.
Err:Raise Number:=*number*	Declanşează eroarea de execuție având numărul specificat. Atunci când o asemenea instrucțiune este executată în interiorul rutinei de eroare, Visual Basic caută lista de apeluri pentru altă rutină de tratare a erorilor. (Lista de apeluri este lanțul de proceduri apelate pentru a ajunge în punctul curent de execuție. Informații suplimentare se găsesc în secțiunea "Ierarhia tratării erorilor.)

Diferența dintre Resume şi Resume Next

Diferența este că **Resume** continuă execuția cu instrucțiunea care a generat eroarea (instrucțiunea este reexecutată), în timp ce **Resume Next** continuă cu instrucțiunea care urmează celei care a generat eroarea. În general, se utilizează **Resume** atunci când eroarea poate fi corectată. Se poate scrie cod care să nu arate utilizatorului căa avut loc o eroare, sau cod care să permită utilizatorului să corecteze eroarea.

Exemplul următor încearcă efectuarea unei împărțiri "sigure", fără a afişa erorile care pot să apară: "Division by zero" (numitor egal cu zero, numărător nenul), "Overflow" (şi numitorul şi numărătorul sunt egali cu zero) sau "Illegal procedure call" (un operand este o valoare nenumerică, sau nu poate fi considerat numeric). In toate cele trei cazuri procedura următoare returnează **Null.**

```
Function Divide (numer, denom) As Variant
         Const mnErrDivByZero = 11, mnErrOverFlow = 6, mnErrBadCall = 5
         On Error GoTo MathHandler
                  Divide = numer / denom
                  Exit Function
```

```
MathHandler:
          If Err.Number = mnErrDivByZero Or Err.Number = mnErrOverFlow _
                                                Or Err.Number = mnErrBadCall Then
                    Divide = Null
          Else
                    MsgBox "Unanticipated error " & Err.Number & ": " & _
                              Err.Description, vbExclamation
          End If
          Resume Next
End Function
```

Reluarea execuției la o linie specificată

Prin **Resume** *line* se dă controlul execuției la linia specificată. Următorul exemplu ilustrează utilizarea acestei instrucțiuni într-o variantă a procedurii FileExists prezentată într-un exemplu anterior.

```
Function VerifyFile As String
          Const mnErrBadFileName = 52, mnErrDriveDoorOpen = 71
          Const mnErrDeviceUnavailable = 68, mnErrInvalidFileName = 64
          Dim strPrompt As String, strMsg As String, strFileSpec As String
          strPrompt = "Enter file specification to check:"
StartHere:
          strFileSpec = "*.*"
          strMsg = strMsg & vbCRLF & strPrompt
          ' Let the user modify the default
          strFileSpec = inputBox(strMsg, "File Search", strFileSpec, 100, _
                                                100)
          ' Exit if user deletes default
          If strFileSpec = "" Then Exit Function
          On Error GoTo Handler
                    VerifyFile = Dir(FileSpec)
                    Exit Function
Handler:
          Select Case Err.Number
                    Case ErrInvalidFileName, ErrBadFileName
                              strMsg = "Your file specification was invalid; _
                                                      try another."
                    Case mnErrDriveDoorOpen
                              strMsg = "Close the disk drive door and try again."
                    Case mnErrDeviceUnavailable
                              strMsg = "The drive you specified was not found. _
                                        Try again."
                    Case Else
                              Dim intErrNum As Integer
                              intErrNum = Err.Number
                              Err.Clear
                              Err.Raise Number:=intErrNum ' Regenerate the error
          End Select
          Resume StartHere ' The user can try another file name
End Function
```

Dacă este găsit un fişier care se potriveşte specificaţiei, funcţia returnează numele fişierului. Dacă nu se potriveşte nici un fişier, funcţia returnează un şir de lungime zero. În cazul apariţiei unei dintre erorile anticipate, se afişează mesajul corespunzător (prin intermediul variabilei strMsg) şi utilizatorul poate încerca un nou nume prin reluarea de la eticheta StartHere.

Dacă apare o eroare diferită de cele anticipate (şi posibile în operaţiunea efectuată), pe ramura de program **Case Else** se regenerează eroarea încât următoarea capcană de erori din lista de apeluri poate să intre în acţiune. Acest fapt este necesar din cauză că dacă eroarea nu ar fi regenerată, codul ar continua să se execute la linia **Resume StartHere.** Prin regenerare, eroarea apare din nou şi noua eroare va fi captată la următorul nivel din stiva de apeluri.

Ierarhia de tratare a erorilor

Un error-handler permis este unul care a fost activat prin executarea unei instrucţiuni **On Error** şi nu a fost interzis (prin **On Error GoTo 0** sau prin părăsirea procedurii care îl conţine). Un error-handler activ este cel care se execută curent. Pentru a fi activ, o secvenţă error-handling trebuie să fie mai întâi permisă, dar nu toate secvenţele permise sunt şi active. De exemplu, după o instrucţiune **Resume**, un handler este dezactivat dar rămâne permis.

La apariţia unei erori într-o procedură, care nu are o secvenţă error-handling permisă, sau într-o rutină error-handling activă, Visual Basic caută lista de apeluri pentru a găsi o altă rutină permisă de tratare a erorilor. Lista de apeluri este secvenţa de apeluri care conduce la procedura curentă; ea este afişată în dialogul **Call Stack**, disponibil în modul break prin meniul **View,** comanda **Call Stack.**

Căutarea în lista de apeluri

Presupunem următoarea succesiune de apeluri:

1. O procedură eveniment apelează procedura A.

2. Procedura A cheamă procedura B.

3. Procedura B apelează procedura C.

În timp ce se execută C, celelalte proceduri sunt suspendate. Dacă apare o eroare în procedura C iar procedura nu are un error- handler, Visual Basic caută înapoi în lista de apeluri - mai întâi B, apoi A, apoi procedura de eveniment, dar atât - şi execută prima secvenţă întâlnită de tratare a erorii. Dacă nu există nici o asemenea secvenţă, atunci se afişează un mesaj implicit de eroare şi se opreşte execuţia aplicaţiei.

Dacă Visual Basic găseşte o secvenţă error-handling, execuţia continuă în acea rutină, ca şi cum eroarea ar fi apărut în procedura care conţine rutina error handling respectivă. Dacă o instrucţiune **Resume** sau **Resume Next** este executată în rutina error handling, execuţia continuă după cum este arătat în tabelul următor.

Instrucţiunea	Rezultatul
Resume	Apelul la procedura pe care Visual Basic tocmai a cercetat-o în lista de apeluri este reexecutat. În apelurile considerate mai sus, dacă procedura A are un error handler permis care include o instrucţiune **Resume**, Visual Basic reexecută apelul la procedura B.
Resume Next	Execuţia se întoarce la instrucţiunea care urmează ultimei instrucţiuni executate în acea procedură. Aceasta este instrucţiunea care urmează apelului la procedura pe care Visual Basic tocmai a cercetat-o în lista de apeluri. În lista de apeluri exemplificată, dacă procedura A are un error handler permis care include o instrucţiune **Resume Next**, execuţia se întoarce la instrucţiunea de după apelul la procedura B.

De notat că instrucţiunea executată este în procedura unde s-a găsit rutina de tratare a erorilor şi nu în mod necesar în procedura unde a apărut eroarea. Dacă nu se ţine seama de acest aspect esenţial al tratării erorilor, execuţia poate părea de neexplicat în momentul apariţiei unei erori. Pentru o verificare mai simplă a codului, se poate trece în modul break în momentul apariţiei unei erori, după cum este explicat ulterior în acest capitol ("Oprirea tratării erorilor").

Dacă erorile tratate în secvenţa error-handler nu includ eroarea apărută, se poate produce o eroare neanticipată în procedura care are un error handler permis: procedura poate să bucleze la infinit, în special când error handlerul execută o instrucţiune **Resume**. Pentru a preveni o asemenea situaţie, se va utiliza metoda **Raise** a obiectului **Err** într-o secvenţă **Case Else** (sau similară) din handler. Aceasta generează o eroare din secvenţa de tratare activă, forţând Visual Basic să caute în lista de apeluri un handler care să se ocupe de eroarea apărută.

Efectul unei căutări regresive în lista de apeluri este greu de prevăzut, deoarece depinde de instrucţiunea **Resume** sau **Resume Next** executată. Trebuie gândit că execuţia nu se reia, în mod necesar, în procedura unde a apărut eroarea, ci în procedura unde este un error-handler activ.

Indicaţii pentru o tratare complexă a erorilor

La scrierea unei aplicaţii Visual Basic mari, care utilizează diverse module, codul de tratare a erorilor poate fi foarte complex. Se vor urmări următoarele linii generale:

- În timpul verificării/depanării codului, se va utiliza metoda **Raise** a obiectului **Err** pentru a regenera eroarea în toate secvenţele de eroare pentru cazurile neanticipate. Pe lângă o căutare ierarhică pentru un handler care să trateze eroarea, Visual Basic va afişa un mesaj de eroare pentru acele erori care nu sunt tratate în cod. Se pot identifica astfel erori posibile şi care pot fi considerate la o nouă dezvoltare a aplicaţiei.
- Se va utiliza metoda **Clear** dacă trebuie să se iniţializeze în mod explicit obiectul **Err** după tratarea unei erori. Acest fapt este necesar atunci când se utilizează tratarea pe loc a

erorilor cu **On Error Resume Next**. Visual Basic apelează automat metoda **Clear** de fiecare dată când execută orice tip de instrucțiune **Resume**, **Exit Sub, Exit Function, Exit Property** sau orice instrucțiune **On Error**.

- Dacă nu se doreşte ca altă procedură din lista de apeluri să capteze eroarea, se va utiliza instrucțiunea **Stop** pentru a forța terminarea codului. Instrucțiunea **Stop** permite examinarea contextului erorii în mediul de dezvoltare.
- Se va scrie o procedură error handler de siguranță (*fail-safe*) sigură care să fie apelată de toate secvențele de tratare a erorilor ca ultimă acțiune pentru erorile care nu sunt gestionate. Această procedură poate efectua o terminare controlată a aplicației prin descărcarea formelor şi salvarea datelor.

Testarea tratării erorilor prin generarea de erori

Simularea erorilor este utilă în timpul verificării aplicației sau când se doreste tratarea unei condiții particulare ca şi cum ar fi echivalentă unei erori de execuție. De exemplu, se scrie un modul care utilizează un obiect definit într-o aplicație externă şi se doreşte ca erorile returnate de acel obiect să fie tratate de restul aplicației precum erorile Visual Basic.

Pentru a testa toate erorile posibile se vor genera erorile prin cod. Această operațiune implică metoda **Raise** a obiectului **Err.** Metoda are sintaxa

object.**Raise** number, source, description, helpfile, helpcontext

unde

object este obiectul **Err**.

number este un întreg Long identificând natura erorii. Erorile Visual Basic (atât Visual Basic cât şi definite de utilizator) sunt în domeniul 0-65535. La stabilirea proprietății **Number** pentru coduri proprii de eroare în module clasă se va adăuga numărul de cod propriu la constanta **vbObjectError**. De exemplu, generarea erorii cu numărul 1050 implică atribuirea valorii vbObjectError + 1050 la proprietatea **Number**.

source este opțional, şir denumind obiectul sau aplicația care generează eroarea. La stabilirea acestei proprietăți pentru un obiect se va utiliza *project.class*. Dacă nu se precizează sursa se utilizează ID-ul proiectului Visual Basic curent.

description este optional, şir care descrie eroarea. În cazul când nu se specifică acest argument se atribuie şirul implicit pentru o eroare Visual Basic (cum este returnat de funcția Error) sau "Application-defined or object-defined error" în cazul unei cod propriu de eroare.

helpfile este optional, calea completă către un fişier Microsoft Windows Help în care se poate găsi ajutor pentru eroarea generată. Implicit se va utiliza informația asociată cu fişierul Visual Basic Help.

helpcontext este optional, ID de context pentru intrarea din fişierul de ajutor. În mod implicit se consideră informația unei erori Visual Basic, dacă există.

În cazul utilizării metodei **Raise** fără specificarea anumitor argumente iar valorile proprietăților obiectului **Err** conțin valori care nu au fost curățate, aceste valori sunt luate în considerare.

Este de menționat că instrucțiunea **Error** poate fi utilizată de asemenea pentru generarea de erori, dar obiectul **Err** dispune de o informație mai bogată. Acest fapt este util în special la crearea modulelor clasă.

Se poate simula orice eroare Visual Basic prin considerarea numărului de cod al erorii respective.

Definirea erorilor proprii

În afara erorilor definite de Visual Basic, utilizatorul poate să definească erori proprii, care să permită identificarea unor condiții proprii aplicației proiectate, captarea acestor erori efectuându-se mai departe după procedura generală. Procesul de definire se realizează prin adăugarea de noi numere de eroare la constanta **vbObjectError**.

Constanta **vbObjectError** rezervă numere de la offsetul propriu la suma offsetului cu 512. Utilizând un număr mai mare decât acesta asigură că numerele proprii de eroare nu intră în conflict cu viitoarele versiuni de Visual Basic.

Pentru a defini numere de eroare proprii se adaugă constante la secțiunea de declarații a modulului:

```
' Error constants
Const gLostCarrier = 1 + vbObjectError + 512
Const gNoDialTone = 2 + vbObjectError + 512
```

Se poate utiliza ulterior metoda **Raise** pentru noile numere de eroare, cu utilizarea eventuală a descrierilor proprii de eroare.

Tratarea inline a erorilor

Atunci când se testează apariția erorilor imediat după fiecare linie de cod care poate produce erori, se zice că tratarea erorilor este *inline*. Utilizând acest mod de tratare a erorilor, se pot scrie funcții şi instrucțiuni care returnează numerele de eroare apărute; se poate genera o eroare Visual Basic într-o procedură şi gestiona eroarea în procedura apelantă; se poate scrie o funcție care să returneze o dată de tip **Variant** şi utiliza acest rezultat în procedura apelantă.

Returnarea numerelor de eroare

Cea mai simplă metodă de acest gen este crearea unor funcții şi instrucțiuni care returnează, atunci când apar erori, numere de eroare (sau constante cu această semnificație) în loc de valorile uzuale. De exemplu

```
Function FileExists (p As String) As Long
          If Dir (p) <> "" Then
                    FileExists = conSuccess
          Else
                    FileExists = conFailure
          End If
End Function
```

care poate fi utilizată prin

```
Dim ResultValue As long
ResultValue = FileExists ("C:\Testfile.txt")
If ResultValue = conFailure Then
          ... ' se trateaază eroarea
Else
          ... ' se procesează fişierul
End If
```

Lucrul esențial în tratarea inline a erorilor este testarea apariției unei erori imediat după instrucțiunea sau apelul de funcție care poate să producă eroarea. În acest mod se poate proiecta un handler care anticipează exact tipul de erori care pot să apară. Această abordare nu necesită ca o eroare de execuție să apară efectiv.

Tratarea erorilor în procedura apelantă

Altă metodă pentru a indica apariția unei erori este să se genereze o eroare Visual Basic în procedură activă şi să se trateze eroarea într-un handler inline din procedura apelantă.

Exemplul următor arată o variantă a procedurii FileExists în care se generează un număr de eroare în cazul eşecului. Înaintea apelului funcției, instrucțiunea **On Error Resume Next** stabileşte proprietățile obiectului **Err** dar nu precizează o rutină error-handling.

Instrucțiunea **On Error Resume Next** este urmată de codul de tratare a erorii. Acesta testează proprietățile obiectului **Err** pentru a vedea dacă a apărut o eroare. Dacă **Err.Number** nu conține zero, atunci a apărut o eroare.

```
Function FileExists (p As String)
          If Dir (p) <> "" Then
                    Err.Raise conSuccess
          Else
                    Err.Raise conFailure
          End If
End Function

Dim ResultValue As Long
On Error Resume Next
ResultValue = FileExists ("C:\Testfile.txt")
If Err.Number = conFailure Then
          ... ' se tratează eroarea
Else
          ... ' se continuă programul
End If
```

Următorul exemplu, mai complex, utilizează atât valoarea returnată, cât şi un argument pentru a detecta apariția unei erori.

```
Function Power (X As Long, P As Integer, ByRef Rezult As Long) _
                              As Long
          On Error GoTo ErrorHandler
          Result = X^P
          Exit Function
ErrorHandler:
          Power = conFailure
End Function

' Apelul funcției Power
```

```
Dim lngReturnValue As Long, lngErrorMayBe As Long
lngErrorMayBe = Power (10, 2, lngReturnValue)
If lngErrorMayBe = conFailure Then
          ... ' tratarea erorii
Else
          ... ' continuarea programului
End If
```

Utilizarea datelor de tip Variant

Altă cale pentru a returna informații despre erorile eventuale este utilizarea datelor de tip Variant şi funcțiile asociate, potrivit exemplului următor. O variabilă de tip Variant are un indicator privind tipul de dată conținută şi acest tag poate fi fixat ca un cod de eroare Visual Basic.

```
Function Power (X As Long, P As Integer) As Variant
          On Error GoTo ErrorHandler
          Result = X^P
          Exit Function
ErrorHandler:
          Power = CVErr(Err.Number)             ' convertește codul de eroare în tagged Variant
End Function

' Apelul funcției Power
Dim varReturnValue As Variant
varReturnValue = Power (10, 2)
If IsError (varReturnValue) Then
          ... ' tratarea erorii
Else
          ... ' continuarea programului
End If
```

Tratarea centralizată a erorilor

Atunci când se adaugă cod de tratare a erorilor la o aplicație, tratarea inline produce tratări repetate ale aceloraşi erori. Codul poate fi redus prin scrierea unor proceduri apelate repetat de secvențele error handling.

Funcția FileErrors din exemplul următor afişează un mesaj adecvat erorii produse şi, când este posibil, permite acoperirea erorii. Este returnat procedurii apelante un cod cu acțiunea de urmat. Este de notat că, undeva în program, se vor defini constantele utilizate.

```
Function FileErrors () As Integer
          Dim intMsgType As Integer, strMsg As String
          Dim intResponse As Integer
          ' Valoarea returnată Înțeles
          ' 0 Resume
          ' 1 Resume Next
          ' 2 Eroare netratabilă
          ' 3 Eroare necunoscută
          intMsgType = vbExclamation
          Select Case Err.Number
                    Case mnErrDeviceUnavailable  ' error 68
                              strMsg = " That device appears unavailable."
                              intMsgType = vbExclamation + 4
                    Case mnErrDiskNotReady       'error 71
                              strMsg = "Insert a disk in the drive and close the door."
                    Case mnErrDeviceIO 'error 57
                              strMsg = " Internal disk error."
                              intMsgType = vbExclamation + 4
                    Case mnErrDiskFull ' error 61
                              strMsg = " Disk is full. Continue?"
                              intMsgType = vbExclamation + 3
                    Case mnErrBadFileName, mnErrBadFileNameOrNumber ' error 64, 52
                              strMsg = " That filename is illegal."
                    Case mnErrPathDoesNotExist   ' error 76
                              strMsg = " That path doesn't exist."
                    Case mnErrBadFileMode        ' error 54
                              strMsg = " Can't open your file for that type of access."
                    Case mnErrFileAlreadyOpen    ' error 55
                              strMsg = " This file is already open."
                    Case mnErrInputPastEndOfFile ' error 62
                              strMsg = " This file has a nonstandard end-of-file marker,"
                              strMsg = strMsg & "or an attempt was made to read beyond"
                              strMsg = strMsg & "the end-of-file marker."
                    Case Else
                              FileErrors = 3
                              Exit Function
          End Select
          intResponse = MsgBox (strMsg, intMsgType, "Disk Error")
          Select Case intResponse
                    Case 1, 4 ' butoanele OK, Retry
                              FileErrors = 0
                    Case 5    ' butonul Ignore
```

```
                        FileErrors = 1
                Case 2, 3 ' butoanele Cancel, End
                        FileErrors = 2
                Case Else
                        FileErrors = 3
        End Select
End Function
```

Această procedură tratează erorile comune legate de fişiere şi unitatea de dischetă. Pentru celelalte erori se returnează valoarea 3. Procedura apelantă va trebui apoi să trateze această eroare, să o genereze încă o dată prin metoda **Raise** sau să cheme altă procedură care să trateze eroarea neanticipată.

Inhibarea tratării erorilor

Dacă o capcană de erori a fost permisă într-o procedură, aceasta este scoasă automat la părăsirea procedurii. Sunt situaţii când este necesară anularea capcanei înainte de părăsirea procedurii gazdă. Acest fapt este realizat de instrucţiunea **On Error GoTo 0**. După executarea acestei instrucţiuni, Visual Basic detectează erorile dar nu sunt captate în procedură. Instrucţiunea poate fi utilizată oriunde, inclusiv în interiorul unei rutine error handler.

Depanarea programelor care au rutine de tratare a erorilor

Atunci când se depanează programe, analiza comportării poate fi complicată în cazul captării erorilor de către rutinele error handler. Posibilitatea de a transforma în comentarii liniile **On Error** din fiecare modul simplifică analiza dar este stânjenitoare.

Este de preferat modalitatea prin care tratarea erorilor este inhibată şi, de fiecare dată când apare o eroare, se intră în modul break. Pentru aceasta, se va selecta opţiunea **Break on All Errors** din fişa **General** a dialogului **Options** (meniul **Tools**). Selectarea acestei opţiuni are ca efect, la apariţia unei erori, intrarea în modul break şi afişarea ferestrei **Watch** cu codul care a produs eroarea.

Tratarea erorilor din obiectele referite

În procedurile care fac referinţă la unul sau mai multe obiecte este mult mai dificil să se determine unde apare o eroare, mai ales dacă eroarea apare într-un obiect al altei aplicaţii. De exemplu, să considerăm o aplicaţie care constă dintr-un modul formă (MyForm), care face referinţă la un modul clasă (MyClassA), care face referinţă la un obiect Microsoft Excel **Worksheet**.

Dacă obiectul **Worksheet** nu rezolvă o eroare particulară apărută în foaia de calcul ci o regenerează, Visual Basic va trece eroarea către obiectul care se referă la obiectul **Worksheet**., adică MyClassA. Visual Basic remapează în mod automat erorile netratate apărute în obiecte din afara Visual Basic către codul de eroare 440.

Obiectul MyClassA poate fie să trateze eroarea (ceea ce este preferabil), fie să o regenereze. Interfaţa specifică faptul că orice obiect care regenerează o eroare apărută într-un obiect referit nu poate să propage pur şi simplu eroarea (să transmită codul de eroare 440), ci trebuie să o remapeze la un număr de eroare cu semnificaţie definită. Acest număr poate fi un număr Visual Basic (dacă se poate efectua o asemenea echivalare), sau un număr propriu, adăugat după procedura explicată anterior.

De câte ori este posibil, un modul clasă va trebui să trateze orice eroare care apare în interiorul modulului şi orice eroare care apr în obiecte referite şi nerezolvate de obiectele însele. Există totuşi erori care nu pot fi tratate din cauză că nu pot fi anticipate. Există şi cazuri când este mai potrivită o tratare a erorii de către obiectul care face referinţa decât de obiectul referit.

Atunci când apare o eroare în modulul unei forme, Visual Basic generează unul dintre numerele de eroare predefinite.

Observaţie. Dacă se creează o clasă publică, trebuie ca fiecare secvenţă care tratează o eroare non-Visual Basic să fie bine documentată. Alţi programatori care se referă la această clasă trebuie să cunoască cum să trateze erorile transmise de obiectul respectiv.

La regenerarea unei erori, celelalte proprietăţi ale obiectului **Err** se vor lăsa neschimbate. Dacă eroarea transmisă nu este tratată, proprietăţile **Source** şi **Description** pot fi afişate pentru a ajuta utilizatorul să ia măsuri corective.

Tratarea erorilor transmise din obiecte referite

Un modul clasă ar putea să includă următoarea secvenţă pentru tratarea erorilor pe care le poate aborda şi regenerarea celor care nu pot fi rezolvate. Secvenţa este explicată după prezentarea codului.

```
MyServerHandler:
        Select Case ErrNum
                Case 7            ' eroarea out-of-memory
                ...
                Case 440  ' erori din obiecte externe
```

```
                        Err.Raise Number:=vbObjectError + 9999
                ' eroare din alt obiect Visual Basic
                Case Is > vbObjectError And Is < vbObjectError + 65536
                        ObjectError = ErrNum
                        Select Case ObjectError
                                ' acest obiect tratează erorile pe baza
                                ' documentaţiei obiectului de unde provine eroarea
                                Case vbObjectError + 10
                                ...
                                Case Else
                                        ' remaparea erorii ca o eroare obiect generică şi
regenerarea ei
                                        Err.Raise Number:=vbObjectError + 9999
                        End Select
                Case Else
                        ' remaparea erorii ca o eroare obiect generică şi regenerarea ei
                        Err.Raise Number:=vbObjectError + 9999
        End Select
        Err.Clear
        Resume Next
```

Instrucţiunea Case 440 captează erorile care apar într-un obiect referit extern aplicaţiei Visual Basic. În acest exemplu, eroarea este propagată utilizând valoarea 9999, deoarece este dificil pentru acest tip de handler central să determine cauza erorii. Apariţia unei astfel de erori este, de obicei, rezultatul unei erori fatale de automatizare (o eroare care produce oprirea execuţiei componentei) sau din cauza unui obiect care nu a tratat corect o eroare captată. Eroarea 440 nu ar trebui propagată decât în cazul unei erori fatale. Dacă această secvenţă ar fi scrisă pentru un handler inline (vezi secţiunea "Tratarea inline a erorilor"), ar fi posibil să se determine cauza erorii şi să se corecteze.

Instrucţiunea Case Is > vbObjectError And Is < vbObjectError + 65536 captează erorile care au originea într-un obiect din aplicaţia Visual Basic, sau din obiectul care conţine handlerul. Numai asemenea obiecte pot produce erori în domeniul specificat.

Documentaţia pentru codul de eroare oferită pentru obiect trebuie să definească codurile posibile, semnificaţia lor, astfel încât această porţiune a handlerului să poată fi scrisă adecvat erorilor anticipate. Codurile de eroare efective pot fi documentate fără offsetul **vbObjectError** sau pot fi documentate după ce s-a adăugat deplasarea, în care caz instrucţiunea **Case Else** trebuie să scadă deplasarea **vbObjectError**. Pe de altă parte, erorile obiectelor pot fi constante, arătate în biblioteca de tipuri a obiectului şi vizibile în Object Browser. În acest caz se utilizează constanta de eroare în **Case Else** şi nu codul de eroare.

Orice eroare care nu este tratată trebuie să fie regenerată cu un nou număr, după cum este în instrucţiunea **Case Else**. În aplicaţia proprie se poate proiecta un handler care să anticipeze acest nou număr definit. Dacă aplicaţia este o clasă publică, trebuie să fie inclusă în documentaţie o explicaţie a acestei noi erori.

Ultima instrucţiune **Case Else** captează şi regenerează orice altă eroare care nu a fost tratată anterior. Deoarece această parte a capcanei va prinde erori care au sau nu adăugată deplasarea **vbObjectError**, remaparea se va efectua simplu către un cod generic "unresolved error". Acest cod se va adăuga la **vbObjectError** indicând oricărui handler că această eroare provine dintr-un obiect referit.

Depanarea tratării erorilor din obiecte referite

Deoarece depanarea aplicaţiilor care se referă la obiecte create în Visual Basic sau la clase definită în module clasă este dificilă, se recomandă selectarea opţiunii **Break in Class Module** din fişa **General** a dialogului **Options** (meniul **Tools**). Cu această opţiune selectată, orice eroare dintr-un modul clasă duce la intrarea clasei în modul break a debuggerului, permiţând analiza erorii.

Depanarea programelor

Tehnicile de depanare prezentate în acest capitol utilizează uneltele de analiză oferite de Visual Basic. Mediul de dezvoltare Visual Basic nu poate să identifice sau să fixeze erorile, dar oferă unelte pentru a analiza fluxul execuţiei şi cum se schimbă valorile variabilelor şi proprietăţilor. Se poate considera că uneltele de depanare ajută la examinarea aplicaţiei pentru a înţelege ce se întâmplă şi de ce.

Uneltele de depanare oferite de Visual Basic includ puncte de oprire (*breakpoints*), expresii de oprire (*break expressions*), expresii de urmărire (*watch expressions*), executarea codului pas cu pas (instrucţiune cu instrucţiune sau procedură cu procedură) şi afişarea valorilor variabilelor şi proprietăţilor. Sunt oferite de asemenea posibilităţi speciale cum ar fi editarea codului şi continuarea execuţiei (edit-and-continue), stabilirea următoarei instrucţiuni care să se execute după o eroare şi testarea cu aplicaţia în modul break.

Tipuri de erori

Există trei tipuri de erori care se pot intâlni într-un program.

Erori de compilare. Acestea rezultă din scrierea incorectă a codului. Aceste erori sunt detectate de Visual Basic la compilarea codului.

Erori de execuţie. Acestea apar în timpul execuţiei aplicaţiei (şi sunt detectate de Visual Basic) atunci când o instrucţiune încearcă să efectueze o operaţiune imposibilă (de genul împărţirii prin zero).

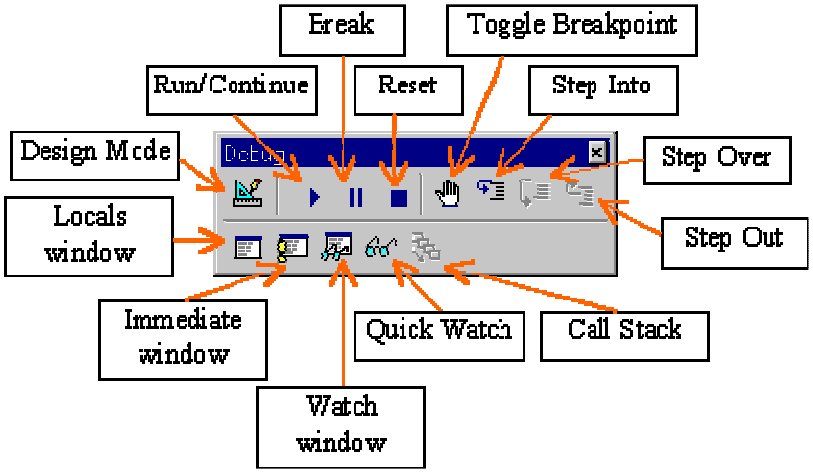
Erori de logică. Acestea apar atunci când aplicaţia nu se execută în modul gândit la proiectare. O aplicaţie poate să fie corctă sintactic, să se execute fără a efectua operaţiuni imposibile şi totuşi să producă rezultate incorecte. Doar testarea aplicaţiei şi analiza rezultatelor poate spune că aplicaţia funcţionează corect.

Unelte de depanare

Nu există artificii magice pentru depanare şi nu există o secvenţă fixă de operaţiuni care să lucreze întotdeauna. În esenţă, ajutorul oferit în depanare este pentru o mai bună înţelegere a mersului aplicaţiei. Se poate astfel fotografia starea aplicaţiei la un moment dat, instantaneul obţinut cuprinzând valori ale variabilelor, expresiilor şi proprietăţilor, precum şi numele apelurilor active de proceduri.

Bara de unelte Debug

Imaginea alăturată arată bara de unelte **Debug** din mediul Visual Basic Editor.



În tabelul următor se explică pe scurt funcţionalitatea uneltelor principale din această bară.

Unealta de depanare	Scop
Run/Continue	Comută din modul design în modul run (**Run**) sau din modul break în modul run (**Continue**).
Break	Comută din run time în modul break.
Reset	Comută din run time sau modul break în modul design.
Toggle Breakpoint	Defineşte o linie într-un modul, unde Visual Basic suspendă execuţia aplicaţiei.
Step Into	Execută următoarea linie executabilă din aplicaţie şi intră în proceduri.
Step Over	Execută următoarea linie executabilă din aplicaţie dar nu intră în proceduri.

Step Out	Execută restul de cod din procedura curentă şi se opreste la linia următoare din procedura apelantă.
Locals Window	Afişează valoarea curentă a variabilelor locale.
Immediate Window	Permite executarea de cod sau interogarea unei valori în timp ce aplicaţia este în break mode.
Watch Window	Afişează valorile unor expresii selectate.
Quick Watch	Listează valoarea curentă a unei expresii în timp ce aplicaţia este în modul break.
Call Stack	În timp ce aplicaţia este în modul break, afişează o cutie de diaolg care arată toate procedurile care au fost apelate dar încă nu s-au executat complet.

Secţiunile următoare aduc explicaţii suplimentare asupra modului de utilizare efectivă a acestor unelte.

Evitarea "bug"-urilor

Respectarea următoarelor reguli poate conduce la evitarea introducerii unor bug-uri în aplicaţie.

Aplicaţia se proiectează cu atenţie prin notarea evenimentelor importante şi a modului în care codul va răspunde la fiecare eveniment. Fiecare procedură generală, ca şi fiecare procedură eveniment trebuie să aibă un scop specific, bine definit.

Se vor include cât mai multe comentarii. La întoarcerea în cod şi analiza comportării aplicaţiei, comentariile privind scopul fiecărei rutine sunt de un real ajutor.

- Se vor utiliza, pe cât posibil, referinţe explicite. Variabilele obiect se declară după cum sunt listate obiectele în cutia **Classes** din Object Browser şi nu prin tipuri **Variant** sau **Object**.

Se va dezvolta sau adapta o schemă consistentă de construcţie a denumirilor pentru variabilele şi obiectele din aplicaţie.

- Se va utiliza declaraţia **Option Explicit** pentru a ocoli erorile de tastare a identificatorilor şi confundarea unui control cu altul.

Design Time, Run Time şi Break Mode

Se utilizează Visual Basic în *design time* pentru a crea o aplicaţie şi în *run time* pentru a o executa. În *break mode* execuţia programului este suspendată astfel încât se pot examina şi modifica date. Bara de titlu Visual Basic arată întotdeauna modul utilizat curent.

Caracteristicile celor trei moduri şi tehnicile pentru comutarea între ele sunt listate în următorul tabel.

Mod	**Descriere**
Design time	Lucrul la crearea unei aplicaţii se execută în acest mod. Se pot proiecta formulare, trasa controale, scrie cod etc. Nu se poate executa cod sau utiliza unelte de depanare, cu excepţia stabilirii de puncte de oprire şi creării de expresii de urmărire. Pentru a trece la run time click pe butonul **Run**. Pentru trecerea în break mode click pe

	Step Into din meniul meniul **Run** şi aplicaţia intră în modul break la prima instrucţiune executabilă.
Run time	În timpul execuţiei, când aplicaţia are controlul, se interacţionează cu aplicaţia ca orice alt utilizator. Se poate vedea codul dar nu se poate schimba. Pentru a trece în design time se acţionează butonul **Reset** iar pentru a trece în break mode click pe butonul **Break**.
Break mode	Execuţia este suspendată în timpul rulării aplicaţiei. Se poate vedea şi edita codul, examina sau modifica date, reporni aplicaţia, sfârşi execuţia sau continua din punctul de oprire. Pentru comutarea în run time, click pe butonul **Continue** (acelaşi cu **Run**). Trecerea în design time se obţine prin butonul **Reset**. Se poat fixa puncte de oprire şi expresii de urmărit în design time, dar celelalte unelte de depanare lucrează doar în modul break.

Utilizarea ferestrelor de depanare

Cu ajutorul ferestrelor de depanare se pot monitoriza valorile expresiilor şi variabilelor în timp ce se parcurg instrucţiunile aplicaţiei. Există trei ferestre de depanare: **Immediate, Watch** şi **Locals**. Pentru afişarea oricăreia dintre aceste ferestre se activează comanda corespunzătoare din meniul **View** sau butonul corespunzător de pe bara **Debug**.

Fereastra **Immediate** arată informaţia care rezultă din instrucţiunile de depanare din cod sau care este cerută prin instrucţiuni scrise direct în fereastră.

Fereastra **Watch** arată valorile curente ale expresiilor urmărite (*watch expressions*), acelea pentru care s-a hotărât monitorizarea. O expresie de oprire (*break expression*) este o expresie santinelă care produce intrarea Visual Basic în modul break atunci când o anumită condiţie definită devine adevărată. În fereastra **Watch**, coloana **Context** indică procedura, modulul, sau modulele în care fiecare expresie santinelă este evaluată. Fereastra **Watch** poate afişa o valoare pentru o expresie urmărită numai dacă instrucţiunea curentă este în contextul specificat; în caz contrar, coloana **Value** afişeazăun mesaj indicând că instrucţiunea nu este în context.

Fereastra **Locals** arată valorile oricărei variabile din domeniul procedurii curente. După cum execuţia trece din procedură în procedură, conţinutul ferestrei **Locals** se modifică pentru a reflecta doar variabilele aplicabile procedurii curente. O variabilă care reprezintă un obiect apare în fereastra **Locals** cu un semn plus (+) în stânga numelui său. Se poate efectua click pe semn pentru a expanda variabila, afişa proprietăţile obiectului şi valorile curente. Dacă o proprietate a obiectului conţine un alt obiect, acesta poate fi expandat de asemenea. Aceeaşi examinare se poate aplica pentru variabilele care conţin tablouri sau tipuri definite de utilizator.

Utilizarea modului break

În design time se poate schimba aspectul sau codul aplicaţiei, dar nu se pot vedea efectele schimbărilor. În run time se poate urmări cum se comportă aplicaţia, dar nu se poate interveni direct în cod.

Modul Break opreşte execuţia aplicaţiei şi oferă un instantaneu al condiţiilor în orice moment. Variabilele şi valorile proprietăţilor sunt păstrate, astfel încât se poate analiza starea curentă a aplicaţiei şi introduce modificări care afectează execuţia aplicaţiei. Când aplicaţia este în modul break se pot efectua următoarele acţiuni:

Modificarea codului aplicaţiei.

Observarea stării interfeţei.

Determinarea apelurilor procedurilor.

Urmărirea valorilor variabilelor, proprietăţilor şi instrucţiunilor.

Schimbarea valorilor variabilelor şi proprietăţilor.

Controlarea fluxului execuţiei prin aflarea/stabilirea instrucţiunii următoare.

Execuţia imediată a unor instrucţiuni Visual Basic.

Controlarea manuală a operării aplicaţiei.

Intrarea în modul break la o instrucţiune cu probleme

În timpul depanării unui program este uneori de dorit oprirea aplicaţiei în acel loc din cod unde se bănuieşte existenţa unei probleme. Acesta este unul dintre motivele pentru care Visual Basic prevede puncte de oprire şi instrucţiuni **Stop**. Un punct de oprire (*breakpoint*) defineşte o instrucţiune sau o mulţime de condiţii unde Visual Basic opreşte execuţia în mod automat şi trece aplicaţia în modul break fără a executa instrucţiunea care conţine punctul de oprire.

Se poate intra în modul break şi manual, prin oricare dintre acţiunile

CTRL+BREAK

- Comanda **Break** din meniul **Run**
- Butonul **Break** de pe bara de unelte **Debug**.

Este posibil să se oprească execuţia atunci când aplicaţia nu are de lucru (*idle* - este între procesarea unor evenimente). Când se întâmplă acest fapt, execuţia nu se opreşte la o linie specifică, dar Visual Basic comută oricum la modul break.

Se poate intra automat în modul break automat atunci când se întâmplă oricare dintre următoarele condiţii:

O instrucţiune generează o eroare de execuţie netratată.

- O instrucţiune generează o eroare de execuţie şi a fost selectat **Break on All Errors** în fişa **General** (meniul **Tools**).
- O expresie de oprire definită în dialogul **Add Watch** s-a modificat sau a devenit True, după cum a fost definită.

Execuţia atinge o linie cu un punct de oprire.

- Execuţia atinge o instrucţiune **Stop**.

Fixarea unei erori run-time şi continuare

Anumite erori din execuţie rezultă din neglijenţa cu care este scris codul. Aceste erori pot fi corectate uşor prin declararea variabilelor, modificarea denumirilor etc. După o asemenea corectură programul poate continua chiar din linia unde s-a oprit. Acest fapt se realizează prin acţionarea comenzii **Continue** (meniul **Run**) sau butonul **Continue**. La continuarea aplicaţiei se poate verifica fixarea erorii.

Dacă se selectează opţiunea **Break on All Errors**, Visual Basic inhibă tratarea erorilor din cod, încât la orice eroare semnalată se intră în modul break. Erorile vor fi captate de secvenţele de tratare doar dacă opţiunea **Break on All Errors** nu este selectată.

Anumite erori, cum ar fi schimbarea declaraţiilor de variabile, adăugarea unor noi variabile etc., necesită repornirea aplicaţiei. În acest caz Visual Basic prezintă un mesaj care lasă repornirea aplicaţiei la voia utilizatorului.

Monitorizarea datelor prin expresii de urmărire

Atunci când un calcul nu produce rezultatul aşteptat sau apar probleme când o anumită proprietate sau variabilă ia o valoare particulară, este de interes să se poată identifica momentul în care o expresie îşi schimbă sau atinge o valoare.

Visual Basic monitorizează în mod automat expresiile definite de utilizator drept expresii santinelă (de urmărit). La trecerea aplicaţiei în modul break, aceste expresii apar în fereastra **Watch**.

Se poate de asemenea instrui o expresie urmărită să treacă aplicația în modul break atunci valoarea ei se modifică sau devine True (pentru condiții). Acest mod de operare reduce, de obicei, parcurgerea pas cu pas a proiectului până când se atinge o anumită condiție (de exemplu, o variabilă de ciclare ajunge la o anumită valoare, sau un indicator dintr-o procedură este modificat).

Adăugarea, editarea sau eliminarea unei expresii urmărite

Toate operațiunile din titlu se pot efectua atât în design time, cât şi în modul break. Pentru a adăuga o expresie santinelă se utilizează dialogul **Add Watch** afişat din meniul **Debug**.

Se utilizează dialogul **Edit Watch**, afişat tot din meniul **Debug**, pentru modificarea sau eliminarea unei expresii definite. Cele două dialoguri, **Add Watch** şi **Edit Watch**, au aceeaşi structură de controale cu excepția butonului **Delete** care apare doar în dialogul **Edit Watch**.

Componentele comune celor două dialoguri sunt prezentate în tabelul următor.

Componenta	**Descriere**
Zona text Expression	Conține expresia pe care expresia urmărită o eveluează. Această expresie este o variabilă, o proprietate, un apel de funcție sau orice altă expresie validă. În dialogul **Add Watch** această boxă de text conține expresia curentă (dacă există).
Grupul de opțiuni Context	Stabileşte domeniul variabilelor urmărite în expresie. Se va completa atunci când există variabile sinonime cu domenii diferite. Se poate restrânge domeniul la o procedură specifică sau la un modul anumit, sau se poate extinde la întreaga aplicație prin selectarea intrărilor All Procedures şi All Modules. Visual Basic poate evalua mai uşor o variabilă într-un context restrâns.
Grupul de opțiuni Watch Type	Stabileşte cum răspunde Visual Basic la expresia urmărită: - urmăreşte expresia şi afişează valoarea în fereastra **Watch** atunci când aplicația intră în modul break; - aplicația intră în modul break automat atunci când se modifică valoarea expresiei sau când aceasta devine True.

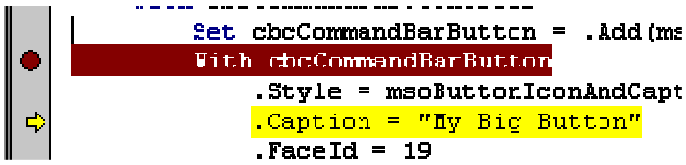
Observație. Se poate adăuga o expresie urmărită prin tragerea expresiei dintr-un modul în fereastra **Watch**.

Utilizarea urmăririi rapide

Când aplicația este în modul break, se poate verifica valoarea unei proprietăți, variabile sau expresii pentru care nu s-a definit în prealabil o expresie de urmărire. Se foloseşte în acest scop dialogul **Quick Watch** (din meniul **Debug** sau de pe bara sinonimă). Dialogul arată valoarea expresiei selectate înrtr-un modul. Pentru a continua urmărirea acestei expresii, se acționează butonul **Add** (inhibat dacă operațiunea de urmărire nu este posibilă); fereastra **Watch** este afişată etc.

Utilizarea unui punct de oprire

În execuție, un punct de oprire (breakpoint) produce oprirea execuției programului exact înaintea unei linii specifice de cod. Atunci Visual Basic execută o procedură şi se ajunge la o linie de cod care are fixat un punct de oprire, atunci aplicația este trecută în modul break.



Se poate stabili sau anula un punct de oprire în modul break, în design time sau în run time dacă aplicaţia este inactivă (idle). Pentru a fixa un punct de oprire se efectuează un click pe marginea din stânga ferestrei modulului, în dreptul instrucţiunii (vezi figura de mai sus). Instrucţiunea astfel marcată este colorată potrivit culorii specificate în dialogul **Options** (meniul **Tools**), fişa **Editor Format**. Prin click pe indicatorul de breakpoint se anulează punctul de oprire respectiv.

Se remarcă în figura anterioară că instrucţiunea curentă este şi ea marcată prin culoare şi un indicator pe latura din stânga a ferestrei de cod. Atunci când se atinge un punct de oprire şi aplicaţia este oprită, se poate examina starea curentă a aplicaţiei, focusul putând fi mutat între forme, module, ferestrele de depanare.

Aplicaţia este oprită exact înaintea executării liniei care conţine punctul de oprire. Dacă se doreşte observarea efectului executării acestei instrucţiunii se va acţiona **Step Into** sau **Step Over**.

Pentru izolarea unei probleme reamintim că o instrucţiune poate să contribuie indirect la eroare datorită atribuirii unor valori incorecte. Examinarea valorilor variabilelor şi proprietăţilor se efectuează în modul break prin ferestrele **Locals**, **Quick Watch,** expresii urmărite şi fereastra **Immediate**.

Utilizarea instrucţiunii Stop

O alternativă la fixarea unui punct de oprire este plasarea unei instrucţiuni **Stop**. La întâlnirea unei instrucţiuni **Stop**, Visual Basic opreşte execuţia şi comută în modul break. Deşi **Stop** acţionează ca un breakpoint, instrucţiunea nu poate fi fixată sau inhibată în acelaşi mod.

Reamintim că o instrucţiune **Stop** opreşte doar temporar execuţia în timp ce o instrucţiune **End** opreşte execuţia, reiniţializează variabilele şi produce întoarcerea la modul design. O aplicaţie oprită prin **Stop** poate fi continuată prin **Continue** din meniul **Run**.

Executarea unor secvenţe de cod selectate

Dacă se poate identifica instrucţiunea care a cauzat eroarea, un singur breakpoint poate localiza problema ivită. Este mai frecventă situaţia în care o întreagă secvenţă de cod este bănuită că produce execuţia incorectă a aplicaţiei. În acest caz, un breakpoint izolează zona de cod şi se parcurge apoi întreaga porţiune pas cu pas prin **Step Into** şi **Step Over**. Dacă este necesar, se poate de asemenea sări peste instrucţiuni sau reîntoarce execuţia la o nouă linie.

Pentru comenzile descrise în tabelul următor trebuie ca aplicaţia să fie în modul break.

Modul de parcurgere	Descriere
Step Into	Execută instrucţiunea curentă şi se opreşte la următoarea linie, chiar dacă ea este în altă procedură.
Step Over	Execută întreaga procedură apelată de linia curentă şi se opreşte la linia următoare liniei curente.
Step Out	Execută restul procedurii curente şi se opreşte la instrucţiunea următoare celei care a apelat procedura curentă.

Utilizarea comenzii Step Into

Se utilizează **Step Into** pentru a executa o singură instrucţiune. Visual Basic trece temporar în run time, execută instrucţiunea curentă şi avansează la următoarea instrucţiune, revine în modul break. Pentru acest tip de operaţiune se acţionează butonul **Step Into** din bara **Debug**.

În cazul scrierii mai multor instrucţiuni pe o aceeaşi linie, Visual Basic le va considera separat (punctele de oprire apar doar la prima instrucţiune din linie).

Utilizarea comenzii Step Over

Comanda **Step Over** se aplică tot instrucțiunii curente, dar aceasta trebuie să conțină un apel la o procedură. Dacă prin comanda **Step Into** s-ar trece în procedură pas cu pas, prin **Step Over** procedura este executată ca o unitate şi se trece apoi la instrucțiunea următoare în procedura curentă. Pentru acest tip de operațiune se acționează butonul **Step Over** din bara **Debug**.

Acest tip de acțiune este util atunci când procedura apelată este verificată. Totuşi, dacă în procedura apelată există o instrucțiune **Stop** sau este definit un punct de oprire, atunci execuția este stopată în acele locuri.

Utilizarea comenzii Step Out

Prin comanda **Step Out** se măreşte viteza de parcurgere a secvențelor de cod în sensul că dacă se ştie că restul instrucțiunilor dintr-o procedură nu pot crea probleme, atunci ele pot fi executate în bloc şi nu pas cu pas. Se revine în procedura apelantă, la instrucțiunea următoare apelului. Pentru acest tip de operațiune se acționează butonul **Step Out** din bara **Debug**.

Trecerea peste secțiuni de cod

În modul break se poate selecta o instrucțiune oarecare, mai departe pe firul execuției, şi click pe **Run To Cursor** din meniul **Debug** permite reluarea execuției din punctul selectat. Se pot astfel sări secvențe neinteresante de cod.

Stabilirea instrucțiunii următoare

O acțiune similară celei descrise în secțiunea anterioară este selectarea unei instrucțiuni oriunde în procedura curentă şi click **Set Next Statement** din meniul **Debug**. Se pot astfel urmări doar anumite fire de execuție sau se pot reexecuta anumite secvențe pentru alte valori date variabilelor/proprietăților.

Indicarea instrucțiunii următoare

Prin click pe **Show Next Statement** din meniul **Debug** se plasează punctul de inserție în linia care urmează a fi executată. Este o modalitate utilă mai ales când se depanează o secvență error handler şi nu se ştie exact care este următoarea instrucțiune. Comanda este disponibilă doar în modul break.

Monitorizarea listei de apeluri (Call Stack)

Dialogul **Call Stack**, afişat prin meniul/bara de unelte **Debug**, arată o listă a apelurilor active de proceduri. Dialogul poate fi afişat doar în modul break. Apelurile active sunt cele care au inițiat proceduri care nu au fost complet executate. Prin lista afişată se poate trasa ordinea de apeluri care au dus la instrucțiunea curentă. Cu cât asemenea lanțuri de apeluri sunt mai lungi, cu atât depanarea aplicației este mai dificilă.

În lista din dialogul **Call Stack** toate apelurile active sunt prezentate într-o serie de apeluri înlănțuite. Ea plasează cea mai timpurie procedură apelată la baza listei şi adaugă apelurile următoare în topul listei. Informația dată pentru fiecare procedură începe cu numele modulului, urmat de numele procedurii. Prin butonul **Show** din dialog se afişează instrucțiunea dintr-o procedură care trece controlul aplicației către următoarea procedură din listă.

Observație. Deoarece dialogul **Call Stack** nu indică variabila asignată unei instanțe a unei clase, nu se poate distinge între instanțe multiple ale claselor.

Testarea datelor şi procedurilor cu fereastra Immediate

În procesul de depanare sau de experimentare a unei aplicații, este adesea necesar să se execute proceduri individuale, să se evalueze expresii sau să se atribuie noi valori variabilelor sau proprietăților. Toate aceste acțiuni pot fi executate din fereastra Immediate. Expresiile sunt evaluate şi rezultatele sunt tipărite în fereastra Immediate.

Tipărirea informațiilor în fereastra Immediate

Există două metode pentru tipărirea în fereastra **Immediate**

- Includerea unor instrucțiuni **Debug.Print** în codul aplicației,
- Introducerea direct în fereastra **Immediate** a instrucțiunilor care utilizează metoda **Print**.

Aceste tehnici oferă următoarele avantaje în raport cu utilizarea expresiilor urmărite:

Aplicația nu trebuie oprită pentru a obține informații. Datele şi mesajele sunt afişate pe măsură ce aplicația se execută.

- Informaţiile sunt afişate într-o fereastră separată (**Immediate**), încât nu se interferează cu ieşirea văzută de utilizatori.

Tipărirea din codul aplicației

Metoda **Print** trimite ieşirea către fereastra **Immediate** atunci când se include calificarea cu obiectul **Debug**. De exemplu

```
Debug.Print "Salary = "; Salary
```

Această tehnică lucrează cel mai bine atunci când există un loc particular în aplicație în care variabila urmărită (aici Salary) se modifică. De exemplu, instrucțiunea anterioară poate aparține unei structuri repetitive.

Tipărirea din fereastra Immediate

Dacă aplicația este în modul break, se poate muta focalizarea în fereastra **Immediate** pentru a examina date. Aici se pot evalua expresii valide, inclusiv expresii care se referă la proprietăți. Modulul activ curent determină domeniul de referință.

Orice instrucțiune care utilizează metoda **Print** poate fi înscrisă şi la terminarea cu ENTER se va evalua expresia şi se va afişa rezultatul. Un semn de întrebare (?) poate înlocui metoda **Print**:

```
? ActiveDocument.Name
```

executată din Microsoft Word, afişează în fereastra **Immediate** numele documentului Word activ.

Atribuirea de valori

În modul break, se pot atribui valori la proprietăți şi variabile. Următorul exemplu prezintă o schemă de utilizare a ferestrei **Immediate** pentru calcule. Fiecare linie (cu excepția celor care afişează rezultatele) se va termina cu ENTER:

```
x = 2
? x+3
5
y = 3
? x*y
6
```

Se poate modifica valoarea unei proprietăți sau a unei variabile din proiect şi relua apoi executarea aplicației. Dacă **Option Explicit** este prezentă în modulul curent, toate variabilele invocate în fereastra **Immediate** trebuie să fie declarate în modul. Domeniul se aplică apelurilor de proceduri ca şi variabilelor.

Testarea procedurilor cu fereastra Immediate

În fereastra **Immediate** se poate evalua orice instrucțiune executabilă validă Visual Basic, dar nu se acceptă declarații de date. Se pot utiliza apeluri la proceduri care permit astfel testarea procedurii cu diverse seturi de argumente.

```
X = Quadratic(2,8,8)
DisplayGraph 50, Arr1
Form_MouseDown 1,0,100,100
```

La apăsarea tastei ENTER, Visual Basic comută la run time pentru a executa instrucțiunea şi se întoarce apoi în modul break. În acest moment se pot vedea rezultatele şi efectele asupra variabilelor şi proprietăților.

Se poate apela orice procedură din forma activă curent, ca şi orice procedură dintr-un modul, cu excepția cazului în care procedura este **Private**, caz în care se poate apela doar în timpul executării modulului.

O procedură poate fi executată în mod repetat, fiecare apel este menținut de Visual Basic ca o instanță separată. Aceasta permite testarea separată a diferitelor seturi de argumente. Dialogul **Call Stack** menține o listă a procedurilor executate de fiecare comandă din fereastra **Immediate**. Se poate prin urmare utiliza **Call Stack** pentru a selecta orice instanță a procedurii şi afişa apoi valorile variabilelor din procedură.

Observație. Deşi cele mai multe instrucțiuni sunt suportate de fereastra **Immediate,** o structură de control este permisă doar dacă poate fi exprimată complet pe o linie: se utilizează caracterul ":" pentru separarea instrucțiunilor care alcătuiesc structura de control.

Verificarea numerelor de eroare

Se poate utiliza fereastra **Immediate** pentru a afişa mesajul asociat cu un număr specific de eroare. De exemplu, tastarea instrucțiunii

```
Error 58
```

produce o casetă cu mesajul asociat erorii

File already exists

Trucuri utile în fereastra Immediate

După introducerea unei instrucțiuni, aceasta se poate reexecuta prin mutarea punctului de inserție înapoi în instrucțiune şi ENTER.

Înainte de ENTER se poate edita instrucțiunea curentă.

- Se poate utiliza mouse-ul sau săgeţile pentru a naviga în fereastra **Immediate**. Nu se apasă ENTER decât pe instrucțiunea care se execută.

CTRL+HOME mută punctul de inserție la linia de început a ferestrei; CTRL+END îl mută la ultima linie.

Tastele HOME şi END mută punctul de inserție la începutul şi, respectiv, sfârşitul liniei curente.

Considerații speciale

Anumite evenimente care sunt o parte comună cu utilizarea Microsoft Windows pot să ridice probleme speciale pentru depanarea unei aplicații. Este important să fim conştienți de aceste probleme pentru a nu complica procesul de depanare.

Dacă rămânem conştienți de modul în care modul break poate pune evenimentele în dezacord cu ceea ce aplicația aşteaptă, avem o şansă să găsim soluții. În anumite proceduri de evenimente, trebuie săutilizăm instrucțiuni **Debug.Print** pentru a monitoriza valorile variabilelor/proprietăților în loc să utilizăm expresii santinelă sau puncte de oprire.

Oprirea execuției în procedurile evenimentelor MouseDown sau KeyDown

Dacă se opreşte execuția în timpul unei proceduri eveniment **MouseDown**, se poate lăsa butonul mouse-ului sau utiliza mouse-ul pentru alte taskuri. Când se continuă execuția, totuşi, aplicația presupune că butonul mouse-ului este încă apăsat. Nu se va obține evenimentul **MouseUp** până când nu se apasă din nou butonul mouse-ului şi apoi se eliberează.

Când se apasă butonul mouse-ului în run time, se opreşte execuția în evenimentul **MouseDown** dacă aici este un breakpoint. În acest scenariu nu se va obține niciodată evenimentul **MouseUp**. Soluția este eliminarea breakpoint-ului din procedura **MouseDown**.

Dacă execuția se opreşte în timpul procedurii **KeyDown**, se aplică din nou considerațiile anterioare. Dacă există un breakpoint în procedura evenimentului **KeyDown**, nu se va obține niciodată evenimentul **KeyUp**.

Oprirea execuției în procedurile evenimentelor GotFocus şi LostFocus

Dacă se opreşte execuția în procedurile evenimentelor **GotFocus** sau **LostFocus**, temporizarea sistemului de mesaje poate cauza rezultate inconsistente. Pentru aceasta se vor utiliza instrucțiuni **Debug.Print** în locul punctelor de oprire din procedurile care tratează evenimentele **GotFocus** şi **LostFocus**.

Distribuirea soluțiilor Microsoft Excel

În acest capitol se discută probleme privind distribuirea soluțiilor dezvoltate pentru Microsoft Excel . Pe lângă modul efectiv de distribuire sunt reamintite şi operațiuni preparatorii cum ar fi protejarea codului, erori posibile etc.

Pregătirea soluției pentru distribuire

În momentul în care s-a închegat o soluție trebuie să se ia o serie de decizii în vederea distribuirii soluției. Prima decizie este cum să se împacheteze proiectul: ca un simplu document sau ca un add-in (sau template în Word).

Dacă se decide distribuirea ca un add-in, este important să se decidă cum şi când se va încărca: automat sau la cerere.

În sfârşit, trebuie să se decidă asupra modului de protejare a codului şi, evident, asupra unei ultime verificări în ceea ce priveşte corectitudinea codului (de obicei ultimul bug apare chiar înainte de punctul final al proiectului).

Alegerea modului de împachetare a soluției

Atunci când se ascrie o soluție în Visual Basic for Applications, se obține un proiect asociat unui document. La distribuirea soluției trebuie să se decidă dacă utilizatorul trebuie să aibă acces la document şi codul asociat sau numai la cod.

Dacă se doreşte accesul atât la document cât şi la cod, se va distribui documentul – adică document Word, caiet Excel sau prezentare PowerPoint. Acest caz este frecvent la scrierea unei soluții verticale – o soluție foarte specifică pentru un utilizator foarte specific. De exemplu, dacă se creează în Microsoft Word un formular de testare a performanței, atunci atât documentul cât şi codul care îl automatizează trebuie să fie accesibil utilizatorului.

Dacă, pe de altă parte, se doreşte ca numai codul (proiectul) să fie accesibil, atunci se va distribui ca un add-in (Microsoft Excel şi PowerPoint) sau ca un template global (Word).

Observație. Utilizatorii nu au acces la foile de calcul dintr-un add-in deoarece acestea sunt ascunse automat la crearea unui add-in dintr-un caiet. În mod similar, atunci când o prezentare PowerPoint este organizată ca un add-in, slide-urile sale sunt eliminate automat. Pentru o discuție suplimentară se va citi secțiunea "Salvarea soluției ca un Add-in sau Global Template".

Se poate interzice accesul utilizatorului la documentul asociat proiectului, fie din cauză că documentul nu este pentru utilizator, fie pentru că el conține date pe care utilizatorul nu trebuie să le modifice sau să le vadă. Acesta este cazul când soluția conține proceduri care extind şi adaptează setul de opțiuni ale aplicației – proceduri care sunt proiectate să aibă o largă aplicabilitate, independente de un document specific. De exemplu, dacă se scriu proceduri care automatizează sarcini de formatare a foilor de calcul, utilizatorul trebuie să aibă acces la aceste proceduri din orice caiet deschis, dar nu trebuie să acceseze caietul asociat proiectului.

La distribuirea soluției ca un add-in sau template trebuie să se decidă separat dacă se va proteja codul proiectului.

Controlul încărcării unui Add-in sau Global Template

Dacă se decide distribuirea soluției proprii ca un add-in sau template, este normal să se decidă momentul încărcării acestuia. Utilizatorul poate întotdeauna să încarce un add-in utilizând dialogul **Template and Add-Ins** sau **Add-Ins** (meniul **Tools**), dar nu aceasta este soluția cea mai bună. Sa alternativă se poate stabili ca un add-in sau template global să se încarce automat la pornirea aplicației. Sau, pentru a nu încetini pornirea aplicației, se poate alege încărcarea ca răspuns la un eveniment anumit sau prin alegerea unei comenzi. În sfârşit, se poate încărca un add-in sau un template global în mod programatic.

Încărcarea unui Add-In sau Global Template la start

Posibilitățile unui add-in devin accesibile unui utilizator doar după ce add-in-ul este încărcat. De exemplu, când un template global este încărcat, Word mixează meniurile, barele de unelte, intrările AutoText şi macrourile din template cu meniul Word. Din acest motiv se poate intenționa ca încărcarea add-inului sau a template-ului să aibă loc la activarea aplicației Word, Excel sau PowerPoint.

Pentru a încărca add-inuri sau template-uri în mod automat la pornirea aplciației, trebuie să se palseze în folderul Office Startup, localizat de obicei pe calea "Program Files\Microsoft Office\ Office\Startup".

În Microsoft Excel se poate de asemenea palsa un add-in în folderul Xlstart sau în folderul desemnat ca un folder de pornire alternativ. Acest fapt se realizează prin atribuirea unei căi şi a unui nume de folder la cheia

HKEY_CURRENT_USER\Software\Microsoft\Office\8.0\Excel\Microsoft Excel\AltStartup

din Windows registry.

Încărcarea automată a unui Add-In în Microsoft Excel prin comanda OPEN din Windows Registry

În plus față de memorarea unui add-in într-unul dintre folderele de pornire (vezi secțiuena precedentă), se poate utiliza comanda OPEN din Windows Registry pentru a specifica fişierele addÎn care se încarcă automat la pornirea aplicației Excel.

Trebuie să se creeze câte o comandă OPEN în cheia de registru

HKEY_CURRENT_USER\Software\Microsoft\Office\8.0\Excel\Microsoft Excel

pentru fiecare fişier add-in. Pentru cazul mai multor fişiere se utilizează OPEN pentru primul fişier, OPEN1 pentru al doilea, OPEN2 pentru al treilea etc.

Sintaxa pentru comanda OPEN este

OPEN =[/*Switch*] *pathandfilename*

unde

/Switch poate fi /R sau /F. Comutatorul /R deschide fişierul read-only iar /F este utilizat pentru a cere încărcarea caietelor. Microsoft Excel citeste doar atâta informație dintr-un workbook câtă este necesară pentru referirea funcțiilor utilizator. Un exemplu este

HKEY_CURRENT_USER\Software\Microsoft\Office\8.0\Excel\Microsoft Excel\OPEN = /R

"C:\OFFICE\OFFICE\LIBRARY\ANALYSIS\ANALYS32.XLL"

Comanda OPEN este suportată doar din motive de compatibilitate şi este necesară pentru cererea de încărcare a macrourilor. Alternativă recomandată este prezentată ulterior în acest capitol, o dată cu Init Commands şi Init Menus.

Încărcarea automată a unui Add-In în Microsoft PowerPoint utilizând valoarea AutoLoad din Windows Registry

Operaţiunea de încărcare automată a unui add-in prin valorile din Windows Registry se poate efectua şi în PowerPoint, dar valorile fixate sunt altele decât în Excel.

Numele fişierului add-in se va da, în Windows Registry, ca subcheie fie sub HKEY_CURRENT_USER, fie sub HKEY_LOCAL_MACHINE. Valoarea Path a cheii trebuie să fie locaţia add-in-ului. Valoarea AutoLoad se va fixa pe 1 pentru o încărcare automată.

Încărcarea Add-In-urilor care au fost încărcate în sesiunea precedentă

Un add-in încărcat şi înregistrat în sesiunea curentă Excel sau PowerPoint va fi încărcat automat la o nouă pornire a aplicaţiei.

Un template global încărcat în Word rămâne disponibil pentru sesiunea curentă, dar nu este reîncărcat automat la o nouă pornire a aplicaţiei.

Observaţie. Un add-in adăugat listei din dialogul **Add-Ins** (utilizând butonul **Add New**) din Excel sau PowerPoint este înregistrat sub HKEY_CURRENT_USER. În continuare, aceste add-in-uri înregistrate vor fi accesibile doar pentru utilizatorul intrat pe maşină în momentul adăugării.

Încărcarea programatică a unui Add-In sau Global Template

Colecţia **Addins** permite instalarea prin program a unui add-in sau template global.

În Word sau Microsoft Excel se va stabili la True proprietatea **Installed** a obiectului **Addin:**

```
Addins("C:\…\Gallery.dot").Installed = True
```

În PowerPoint, proprietatea **Loaded** a obiectului **Addin** se va stabili la True pentru înregistrarea automată, aceeaşi valoare dându-se proprietăţii **Registered** pentru a înregistra add-in-ul.

```
With Addins("C:\…\mytools.ppa")
          .Loaded = True
          .Registered = True
End With
```

Încărcarea programatică a unui add-in ca răspuns la un eveniment

Dacă se doreşte încărcarea programatică a unui add-in ca răspuns la un eveniment, codul care comandă această operaţiune se plasează în procedura de eveniment.

Deschiderea şi încărcarea programatică a unui template global prin linia de comandă

Se utilizează următoarea linie de comandă, de exemplu, unde *path* este locaţia fişierului Add-in.dot, pentru a instala un add-in numit Add-in.dot, şi executarea unei proceduri numită Main din modulul denumit Add-inInstall:

```
WinWord "<Path>\Add-in.dot" /mAdd-inInstall
```

Word are un număr de argumente pe linia de comandă. Comutatorul /m execută codul din modulul specificat. De notat că nu există nici un spaţiu după "m" şi că nu se specifică procedura Main; doar o procedură denumită Main se va executa, în lipsa unei asemenea proceduri nu se execută nimic la încărcarea modulului. În continuare este un exemplu de procedură Main.

```
' numele modulului este Add-inInstall, pentru a funcţiona cu
' linia de comandă din exemplul anterior
Option Explicit
Sub Main()
'Check to see if Add-In is loaded in Add-In list
          Dim wkbAddin As Word.AddIn
          On Error GoTo ErrorHandler
          Set wkbAddin = AddIns(ThisDocument.Name)
          If wkbAddin Is Nothing Then
                    ' Add the template to the add-ins collection and install it
                    Set wkbAddin = AddIns.Add(ThisDocument.FullName, True)
          End If
          ' If template is the active document close it
          If ThisDocument.Name = ActiveDocument.Name Then
                    ' The Add-in should not be dirty
                    ThisDocument.Close
```

```
        End If
        Exit Sub
Errorhandler:
        ' Only ignore "Subscript out of range" errors
        Select Case Err.Number
                Case 9, 5941
                        Err.Clear
                        Resume Next
                Case Else
                        ' Assert the error when in Debug mode
                        #If DebugMode Then
                                AssertError
                        #End If
                        ' Insert other error handling code here
        End Select
End Sub
```

Descărcarea programatică a unui Add-in sau Template

Un add-in rămâne încărcat până când se termină sesiunea curentă, până la descărcarea prin program sau până la descărcarea prin dialogul **Add-Ins** (sau **Templates and Add-ins**) activat prin meniul **Tools**. Prin descărcarea unui add-in se salvează memoria disponibilă.

În Word sau Microsoft Excel, pentru descărcarea unui add-in sau template global, se stabileşte la False proprietatea **Installed** a obiectului **Addin**.

```
Addins(C:\…\Gallery.dot").Installed = False
```

În PowerPoint, proprietatea **Loaded** a obiectului **Addin** se va stabili pe False pentru descărcare, iar proprietatea **Registered** se va stabili pe False pentru a înlătura înregistrarea add-in-ului.

Încărcarea la cerere

Încărcarea unui add-in sau template la pornirea aplicației poate să crească semnificativ timpul pornirii aplicației. Se poate utiliza atunci încărcarea la cerere pentru a amâna încărcarea până atunci când utilizatorul apelează o comandă sau o procedură din add-in.

Este de menționat că wizard-urile şi add-in-urile inițiate de o comandă din menu sunt încărcate, în mod automat, la cerere.

Există mai multe tehnici pentru a stabili încărcarea la cerere, specifice diferitelor aplicații.

Observație. Dacă se intenționează utilizarea unui add-in sau template ca o bibliotecă de cod, apelată din alte proiecte, nu trebuie ca aceasta să fie încărcată explicit deoarece Visual Basic va încărca add-in-ul sau template-ul referit la cerere, atunci când o procedură din bibliotecă este apelată.

Utilizarea referințelor explicite în Microsoft Excel

Prima metodă de încărcare la cerere este utilizarea unei referințe explicite atunci când se atribuie un nume de macro la o unealtă. Următoarea linie cheamă procedura MySub care stă în modulul ThisWorkbook din caietul (add-in) "C:\My Documents\MyTools.xla":

```
'C:\My Documents\MyTools.xla'!ThisWorkbook.MySub
```

Add-in-ul este încărcat doar atunci când este executat macro-ul.

Pentru soluții care sunt distribuite utilizând un program de instalare se va vedea secțiunea "Adăugarea de meniuri şi submeniuri la Microsoft Excel fără încărcarea unui Add-In".

Încărcarea la cerere a bibliotecilor de funcții în Microsoft Excel

Dacă se distribuie o soluție care conține o bibliotecă de funcții macro ar fi de dorit încărcarea la cerere a acestei biblioteci. De asemenea, în mod normal este de dorit ca funcțiile să fie listate în Function Wizard. Pentru încărcarea la cerere a funcțiilor este nevoie de macro-uri XLM (Microsoft Excel 4.0 macro sheets). Aceste macro-uri sunt de obicei doar "învelişuri" pentru funcții Visual Basic.

Atunci când Microsoft Excel vede că un Add-In este marcat pentru încărcarea la cerere, el citeşte doar anteturile macro-urilor funcții; rațiunea de prezentare sub formă XLM este aceea că informația din anteturi poate fi citită fără a încărca Visual Basic for Applications (care se va încărca doar atunci când este necesar).

Modificarea unui add-in pentru a beneficia de încărcare la cerere

1. Se adaugă la caietul add-in o foaie Microsoft Excel 4.0 macro: click dreapta pe un tab Worksheet şi selectare **Insert**, din dialog se va selecta **Ms Excel 4.0 Macro.**

Crearea unui macro funcție care învelește o funcție Visual Basic.

În prima celulă se scrie numele funcției.

- În celula de sub numele funcţiei se inserează funcţia **Argument(Name_Text, Data_Type_ID).**
- **Name_Text** este numele parametrului. Ideal ar fi ca acesta să fie numele parametrului funcţiei din VBA pe care se presupune că-l reprezintă.
- **Data_Type_ID** este ID-ul care determină tipul de dată pe care Excel îl acceptă pentru acest argument; acest parametru este opţional.

Se repetă paşii b, c şi d pentru fiecare parametru din funcţia VBA.

- Dacă este necesar să se specifice tipul rezultatului se va utiliza funcţia **Result(Data_Type_ID)**.
- Funcţia **Return(value)** semnalează sfânrşitul macro-ului funcţie.

Se selectează toate celulele care conţin macro-ul.

- Se selectează **Define** din submeniul **Name** (meniul **Insert**).

Se denumeşte domeniul cu numele funcţiei.

- Se selectează butonul radio **Function** din grupul **Macro**.
- Click pe **Add** şi apoi **OK**.

1. Se selectează **Insert** din bara de meniu şi **Define** din submeniul **Name**. Se adaugă numele "__DemandLoad" care se referă la "=TRUE" (se tastează exact aşa în boxa **Refers To**). Este de observat că există două caractere "_" în faţa lui DemandLoad. Click **Add**, apoi **Close**.
2. Se stabileşte proprietatea **IsAddin** a obiectului Workbook la TRUE.

Se salvează caietul.

În continuare este un exemplu de macro care înfăşoară o funcţie VBA.

```
MyFunction
=ARGUMENT("szArgOne")
=ARGUMENT("intArgTwo")
=RETURN(vMyFunction(szArgOne, intArgTwo))
```

Funcţia VBA fiind de forma

```
Function vMyFunction (szArgOne As String, intArgTwo As Integer) As String
…
End Function
```

Aceşti paşi permit ca add-in-ul să fie încărcat la cerere. După ce este încărcat, add-in-ul nu este descărcat din memorie până când se termină instanţa curentă a aplicaţiei Excel.

Adăugarea meniurilor în Excel fără încărcarea unui Add-in

Microsoft Excel are două chei noi de registru, care sunt citite la pornirea aplicaţiei şi care creează intrări de meniu oferind puncte de intrare într-un add-in; acest add-in se încarcă doar atunci când este cerut de o comandă dintr-un meniu, care se referă la add-in. Cheile respective sunt **Init Menus** şi **Init Commands**. În continuare se prezintă de asemenea şi cheia **Delete Commands**, care permite eliminarea unor intrări de meniu.

Cheia Init Menus

(HKEY_CURRENT_USER\Software\Microsoft\Office\8.0\Excel\Init Menus) conţine câte o valoare String pentru fiecare menu adăugat unei mare de meniu built-in. Fiecare nume de valoare este unic şi identifică meniul creat. Şirul are sintaxa:

Value_name = Menu_bar_num, Menu_name,Menu_position[, Menu_parent]

Argument	Descriere
Menu_bar_num	Numărul barei de meniu built-in la care se doreşte adăugarea meniului. Barele de meniu se schimbă după tipul foii active. Numerele sunt: 3 Nil menu bar (nu există caiet deschis)

	10 worksheet, dialog sheet, macro sheet 4.0 11 chart sheet 12 modul Visual Basic
Menu_name	Numele noului meniu
Menu_position	Poziţia noului meniu pe bara meniu. Aceasta poate fi numele unui meniu după care se plasează cel nou, sau un număr indicând poziţia noului meniu de la stânga barei meniu.
Menu_parent	Este opţional: dacă se defineşte un submeniu, acesta este numele sau numărul meniului care va conţine noul submeniu, definirea căruia fiind controlată de restul parametrilor.

Cheia Init Comands

(HKEY_CURRENT_USER\Software\Microsoft\Office\8.0\Excel\Init Commands) conţine câte o valoare de tip string pentru fiecare comandă adăugată unui meniu. Fiecare nume de valoare este unic şi identifică comanda adăugată. Şirul are sintaxa

Value_name = Menu_bar_num,Menu_name,Command_name,Macro,

Command_position,[Macro_key] ,[Status_text] ,[Help_reference]

parametrii fiind explicaţi în tabelul următor.

Argument	**Descriere**
Menu_bar_num	Numărul barei de meniu built-in la care se doreşte adăugarea meniului. Barele de meniu se schimbă după tipul foii active. Numerele sunt: 10 Worksheet, dialog sheet, macro sheet 4.0 11 Chart sheet 12 modul Visual Basic
Menu_name	Numele meniului sau submeniului. Submeniurile sunt indicate printr-un şir "meniu\submeniu", caracterul backslash delimitând numele meniului de cel al submeniului. Submeniul trebuie să fie existent deja sau să fie declarat în cheia Init Menu.
Command_name	Numele noii comenzi.
Macro	Referinţa la o procedură dintr-un caiet add-in. Alegerea comenzii deschide add-in-ul şi execută procedura. Procedura va şterge comanda adăugată de această cheie de registru şi o va înlocui cu o comandă care va executa procedura necesară.
Command_position	Poziţia noii comenzi în meniu. Aceasta poate fi numele unei comenzi după care se plasează cea nouă, sau un număr indicând poziţia noii comenzi pe

	meniu. Dacă este omis, comanda apare la sfârşitul meniului.
Macro_key	Este opțional, cheia ataşată procedurii, dacă există.
Status_text	Este opțional, mesajul afişat în bara de stare atunci când se selectează comanda.
Help_reference	Este opțional, numele fişierului şi numărul intrării pentru un Help ataşat comenzii.

Cheia Delete Commands

Se poate utiliza cheia Delete Commands pentru a şterge comenzi din meniurile buil-in. **Add-In Manager** (comanda **Add-Ins** din meniul **Tools**) citeşte şi scrie valori în cheia **Delete Commands**. Cheia conține câte un şir pentru fiecare comandă care este ştearsă din meniu. Fiecare nume de valoare este unic şi identifică comanda eliminată. Şirul are sintaxa

Value_name = Menu_bar_num,Menu_name,Command_position

Argument	**Descriere**
Menu_bar_num	Numărul barei de meniu built-in unde se doreşte modificarea meniului. Barele de meniu se schimbă după tipul foii active. Numerele sunt: 3 Nil menu bar (nu există caiet deschis) 10 worksheet, dialog sheet, macro sheet 4.0 11 chart sheet 12 modul Visual Basic
Menu_name	Numele meniu
Menu_position	Poziția comenzii pe bara meniu. Aceasta poate fi numele unei comenzi sau un număr indicând poziția comenzii în meniu.

Observație. Este recomandabil să nu se elimine comanda **Exit** din meniul **File** dacă nu s-a creat o cale alternativă de părăsire a aplicației Microsoft Excel.

Scrierea codului executat la încărcarea sau descărcarea unui Add-in sau Global Template

Se poate scrie cod care să se execute ca răspuns la încărcarea sau descărcarea unui add-in. De exemplu, atunci când se încarcă un add-in se poate executa o procedură care să afişeze o bară de unelte dând acces la funcțiunile add-in-ului. În mod similar, la descărcarea add-in-ului, aceeaşi bară de unelte se poate elimina prin executarea unei proceduri speciale. Fiecare aplicație are totuşi un mod propriu de realizare a acestor acțiuni.

Microsoft Excel – evenimentele AddinInstall şi AddinUninstall

Fiecare add-in are două evenimente, **AddinInstall** şi **AddinUninstall**, care sunt declanşate la încărcarea şi, respectiv, descărcarea add-in-ului, atât prin dialogul **Add-ins** cât şi programatic. Aceste evenimente sunt locurile ideale de plasare a intrărilor în cheile Init Menus şi Init Commands din Windows Registry.

Se poate utiliza evenimentul **Open** a unui caiet pentru a efectua înregistrarea caietului şi adăugarea lui la dialogul **Add-In.** În următorul exemplu se prezintă o structură generală de utilizare a procedurilor evenimentelor AddinInstall şi AddinUninstall.

```
Option Explicit
Private Sub Workbook_AddinInstall()
         ' Add code to customize the Microsoft Excel User Interface (UI) here
```

```
          MsgBox "Addin Installed"
End Sub
Private Sub Workbook_AddinUninstall()
          ' Add code to remove customization of the Microsoft Excel UI here
          MsgBox "Addin Uninstalled"
End Sub
Private Sub Workbook_Open()
          ' Check to see if add-in is loaded in Microsoft Excel's AddIn list
          Dim wkbAddIn As Excel.AddIn
          On Error GoTo Errorhandler
          Set wkbAddIn = AddIns(ThisWorkbook.Name)
          If wkbAddIn Is Nothing Then
                    ' Setting the CopyFile argument to true will cause the add-in
                    ' to be copied to the local harddisk if it is on a removeable
                    ' medium.
                    Set wkbAddIn = AddIns.Add(ThisWorkbook.FullName, True)
                    ' Thenext line will cause the Workbook.AddInInstalled event
                    ' to fire.
                    wkbAddIn.Installed = True
          End If
          ' Initialize add-in here.
          Exit Sub
Errorhandler:
          ' Only ignore "Subscript out of range" errors.
          If Err.Number = 9 Then
                    Err.Clear
                    Resume Next
          Else
                    ' Assrt the error when in Debug mode.
                    #If DebugMode Then
                              AssertError
                    #End If
                    ' Insert other error handling code here.
          End If
End Sub
```

Verificarea finală a codului

Înaintea împachetării soluției ca un add-in sau un template global, este utilă o ultimă verificare pentru eliminarea unor erori frecvente, amintite în continuare.

Verificarea referințelor la ActiveWorkbook sau ActiveDocument

Reamintim că dacă proiectul VB dintr-un add-in sau template global conține o referință explicită sau implicită la documentul activ sau la caietul activ, el sa ve referi în execuție la acel document sau caiet care se va întâmpla să fie activ în momentul aplicării referinței. Pentru referința la add-in sau template se va utiliza prin urmare **ThisWorkbook** sau **ThisDocument**.

De exemplu, ambele proceduri scrise în continuare apelează foaia denumită "Addin Definition" în caietul care este activ în momentul execuției codului. Prima procedură conține o referință explicită, prin **ActiveWorkbook.** A doua procedură are o referință implicită: deoarece nu se referă explicit la un caiet anumit, referința este presupusă în caietul activ.

```
Sub ExplicitReference()
          Set rMnuTable = ActiveWorkbook.Worksheets("Addin Definition). _
                                        Range("MenuDefinition")
          Add_Menu rMnuTable
End Sub
Sub ImplicitCode()
          Set rMnuTable = Worksheets("Addin Definition). _
                                        Range("MenuDefinition")
          Add_Menu rMnuTable
End Sub
```

Următorul cod utilizează proprietatea **ThisWorkbook** pentru a realiza o referință la caietul în care se execută codul – adică în caietul add-in.

```
Sub CorrectCode()
          Set rMnuTable = ThisWorkbook.Worksheets("Addin Definition). _
                                        Range("MenuDefinition")
          Add_Menu rMnuTable
End Sub
```

Apelul rutinelor din alte proiecte

Dacă se doreşte apelarea unor funcţii, proceduri şi clase care sunt definite în alt proiect, proiectul apelant trebuie să aibă o referință la proiectul apelat. Este de notat că nu se pot crea referințe la prezentări PowerPoint decât în cazul în care acestea sunt salvate ca add-in-uri PowerPoint; după salvarea ca un add-in prezentarea poate fi referită din alte prezentări.

Este evident că pentru a utiliza referinţe la proiecte trebuie ca proiectele implicate să poarte denumiri diferite (de regulă, la începutul utilizării VBA, proiectele se lasă cu denumirile implicite). Numele implicit al unui proiect se poate vedea/modifica fie în fereastra **Properties**, fie în **Project Name** din dialogul **Project Properties** (obţinut prin click dreapta în VBE şi alegerea *project name* Properties).

Referinţa la un alt proiect se realizează manual prin dialogul **References** (meniul **Tools**) sau programatic prin metoda **AddFromFile** a colecţiei **References**. Linia următoare stabileşte o referinţă din caietul activ la proiectul din add-in-ul MyTools.xla:

```
ActiveWorkbook.VBProject.References.AddFromFile "C:\Tools\MyTools.xla"
```

Este însă de notat că înainte de a stabili programamtic o referinţă, trebuie (în Word, Excel, PowerPoint) stabilită o referinţă la biblioteca de tipuri Visual Basic for Applications Extensibility. Această bibliotecă oferă obiectele care permit lucrul programatic cu proiectul VBA.

Observaţie. Rutina Auto_Open dintr-un add-in referit nu este executată la încărcare prin referinţă dintr-un alt add-in sau prezentare.

Evitarea referinţelor nerezolvate

La salvarea şi distribuirea unui add-in sau template, referinţele stabilite pentru proiectul asociat rămân valabile. Totuşi, dacă add-in-urile sau şabloanele referite nu sunt la aceleaşi locaţii pe maşina utilizatorului cum erau pe maşina de dezvoltare, referinţele nu vor putea fi rezolvate la beneficiar şi soluţia nu va fi executată.

În Microsoft Excel, Visual Basic încearcă rezolvarea referinţelor prin căutare în locaţiile:

În acelaşî folder cu add-in-ul sau template-ul apelant.

În folderul rădăcină a aplicaţiei gazdă.

În folderul System şi Windows.

În toate folderele variabilei de mediu Path.

În Microsoft Excel, Visual Basic priveşte de asemenea în folderele adăugate la valoarea **Add-in Path** din registry.

Dacă o referinţă nu este rezolvată după completarea căutării prezentate, execuţia este oprită.

În continuare sunt enumerate câteva metode care pot fi utilizate în evitarea/rezolvarea referinţelor nerezolvate.

Toate fişierele se vor livra în acelaşi folder. Aceasta este soluţia comună.

În Excel se poate actualiza cheia de registry Add-ins Path. Microsoft Excel oferă cheia de registru

HKEY_CURRENT_USER\Software\Microsoft\Office\8.0\Excel\Microsoft Excel\Add-in Path

care punctează către folderele unde Microsoft Excel priveşte pentru add-in-uri. Valoarea Add-in Path este un şir care constă din căi multiple delimitate de caractere ";". La instalarea aplicaţiei Excel, programul Setup adaugă căile către folderele Library, Analysis şi Solver. Alte programe de instalare pot să modifice valoarea Add-in Path. Modificarea trebuie să fie "politicoasă": noile intrări se vor adăuga celor existente. Programele de tip uninstall trebuie să elimine doar porţiunea din valoare adăugată la instalare.

Adăugarea folderului soluţiei la variabila de mediu Path.

Se pot utiliza funcţiile API **GetEnvironmentVariable** şi **SetEnvironmentVariable** pentru adăugarea temporară a folderului soluţiei, sau se poate utiliza scrierea în instrucţiunea Path din Autoexec.bat pentru o adăugare permanentă.

Exemplul următor utilizează funcţiile **GetEnvironmentVariable** şi **SetEnvironmentVariable** pentru a modifica temporar variabila de mediu Path.

```
Option Explicit
Private Declare Function GetEnvironmentVariable Lib "kernel32" _
                  Alias "GetEnvironmentVariableA" (ByVal lpName As String, _
                                                   ByVal lpBuffer As String, _
                                                   ByVal nSize As Long) As Long
Private Declare Function SetEnvironmentVariable Lib "kernel32" _
                  Alias "SetEnvironmentVariableA" (ByVal lpName As String, _
                                                   ByVal lpValue As String) As Long
' AppWord is a user defined compile directive
#If AppWord = True Then
Public Sub AutoExec()
```

```
          AppendEnvironmentPath ThisDocument
End Sub
#End If
'AppExcel is a user defined compile directive
#If AppExcel = True Then
Public Sub Workbook_Open()
          AppendEnvironmentPath ThisWorkbook
End Sub
#End If
Private Sub AppendEnvironmentPath (ThisDocObject As Object)
          Const ENV_PATH As String = "Path"
          Dim iRet As Long
          Dim szPath As String
          SzPath = String(1024, 0)
          ' Retrieve the current environment setting.
          IRet = GetEnvironmentVariable(ENV_PATH, szPath, 1024)
          If iRet Then
                    ' iRet contains the length of the returned string.
                    ' trim any trailing characters
                    szPath = Mid$(szPath, 1, iRet)
                    ' See if the template path is included in the path statement.
                    If InStr(1,szPath, ThisDocObject.Path, vbTextCompare) = 0 Then
                              ' Path is not part of the environment.
                              szPath = szPath & ";" & ThisDocObject.Path
                              iRet = SetEnvironmentVariable (ENV_PATH, szPath)
                              If iRet = 0 Then
                                        ' Handle error here. Template path was not appended to
                                        ' environment.
                                        Err.Raise vbObjectError + Err.LastDllError, _
                                                  ThisDocObject.Name & ".AppendEnvironmentPath", _
                                                  "Path environment was not set."
                              End If
                    End If
          Else
                    ' should raise an error here. This should never happen …
                    Err.Raise vbObjectError + Err.LastDllError, _
                              ThisDocObject.Name & ".AppendEnvironmentPath", _
                              "No path environment was found."
          End If
End Sub
```

- Se poate scrie un add-in agent de referință (reference broker) care să adapteze referințele pentru a maşină specifică. Un asemenea agent de referință ar trebuie să verifica referința şi să lege înainte ca soluția să se execute (sau la momentul încărcării). Acest tip de componentă este bună pentru soluții trietajate (three-tier). Agentul trebuie să lege dinamic componenta front-end de componenta back-end pe baza versiunii, nivelului de securitate sau alți parametri determinați de administrator. Această tehnică poate fi utilizată la actualizarea automată a componentelor. O cale pentru determinarea dacă o referință este validă este utilizarea proprietății **IsBroken** a obiectului **Reference**. Proprietatea **IsBroken** este True dacă referința nu se referă la o intrare validă în Windows Registry.

Apelul unei rutine dintr-un Add-in PowerPoint fără fixarea unei referințe

Dacă se dorește încărcarea temporară a unui fişier add-in şi executarea unei rutine din fişier, se va utiliza o instrucțiune **Application.Run** şi imediat apoi se va descărca fişierul add-in. La executarea **Application.Run**, Microsoft PowerPoint va încărca add-in-ul şi Va executa macroul specificat. Pentru a descărca fişierul se utilizează proprietatea **Loaded,** după modelul:

```
Sub CallAddInMacro()
          Application.Run "…\MyAddIn.ppa!MyMacro"
          AddIns("MyAddIn").Loaded = False
End Sub
```

Ultima instrucțiune poate fi înlocuită şi cu

```
AddIns(AddIns.Count).Loaded = False
```

Apelul rutinelor folosind referințe legate târziu

Se poate apela orice procedură publică, proprietate sau variabilă din modulul **ThisDocument** sau **ThisWorkbook** al unui document sau caiet utilizând o referință târzie la obiectul asociat **Document** sau **Workbook**. De exemplu, următoarele instrucțiuni atribuie obiectul **Document,** returnat de **Open,** unei variabile obiect. După aceea se apelează procedura MySub din modulul clasă **ThisDocument** al documentului:

```
Dim objDoc As Object
Set objDoc = Documents.Open(szFileName)
objDoc.MySub
```

Dacă documentul reprezentat de variabila objDoc nu are, în modulul său **ThisDocument**, o procedură publică denumită MySub, atunci apare o eroare de execuție. În acest mod se poate apela cod doar din module

ThisDocument sau **ThisWorkbook.** Dacă se doreşte apelul unor proceduri din alte module, clase sau forme ale proiectului asociat documentului/caietului deschis, se vor scrie proceduri intermediare în modulul **ThisDocument** sau **ThisWorkbook** pentru a le accesa.

Observaţie. Procedeul descris nu funcţionează în PowerPoint deoarece nu există un modul **ThisPresentation** în proiect.

Protejarea sau neprotejarea codului

Stabilirea proprietăţii **IsAddin** nu protejează codul sursă al proiectului. Pentru protejarea codului se va marca **Lock project for viewing** (meniul **Tools**, project **Properties,** fişa **Protection**) şi se va fixa o parolă. Dacă se fixează doar parola, fără blocarea accesului la cod, acesta poate fi văzut dar nu pot să deschidă dialogul *project* **Properties** fără cunoaşterea parolei. După stabilirea parolei proiectul se va salva obişnuit prin **Save** (meniul **File** din Visual Basic Editor).

Pentru înlăturarea protecţiei se va elimina marcarea boxei de control **Lock project for viewing** şi parola introdusă.

Salvarea soluţiei ca un Add-in sau Global Template

Fiecare aplicaţie din Microsoft Office are un mod propriu de creare a unui add-in. În Microsoft Excel şi PowerPoint se salvează fişierul într-un format specific. În Word se salvează documentul ca un fişier template (.dot).

Crearea unui Add-in în Excel

La transformarea unui caiet într-un add-in, foile caietului sunt ascunse şi procedurile din proiect sunt ascunse utilizatorului (procedurile nu mai apar în dialogul **Macros**).

Pentru crearea unui add-in Excel, se fixează pe True proprietatea **IsAddIn** a caietului care conţine codul. O cale de realizare a acestui lucru este salvarea unei copii a caietului respectiv ca un add-in Excel prin selectarea tipului de fişier Microsoft Excel Add-In (*.xla) în dialogul **Save As.**

Proprietatea poate fi stabilită şi manual din VBE: se selectează "ThisWorkbook" în Project Explorer şi se modifică proprietatea **IsAddIn** în Properties Window. După ce proprietatea **IsAddIn** este fixată pe True, caietul este ascuns în Microsoft Excel.

Proprietatea se poate stabili şi programatic.

Observaţie. Dacă este nevoie să se modifice elementele caietului după crearea add-in-ului, se fixează proprietatea **IsAddIn** la False, caietul devine vizibil şi editabil. După modificare se reface proprietatea **IsAddIn** şi se salvează caietul.

Depanarea unui Add-in sau Global Template

Pentru depanare se utilizează, cu unele particularităţi, deprotejarea codului şi efectuarea acţiunilor uzuale de verificare a codului.

Depanarea unui Add-in Excel

Pentru depanarea unui add-in Excel o dată încărcat, se înlătură protecţia proiectului respectiv. Dacă este necesar să se vadă caietul asociat se fixează proprietatea **IsAddIn** la False.