

IDŹ DO

PRZYKŁADOWY ROZDZIAŁ



SPIS TREŚCI

KATALOG KSIĄŻEK

KATALOG ONLINE

ZAMÓW DRUKOWANY KATALOG

TWÓJ KOSZYK

DODAJ DO KOSZYKA

CENNIK I INFORMACJE

ZAMÓW INFORMACJE
O NOWOŚCIACH

ZAMÓW CENNIK

CZYTELNIA

FRAGMENTY KSIĄŻEK ONLINE

Visual Basic .NET. Opis języka

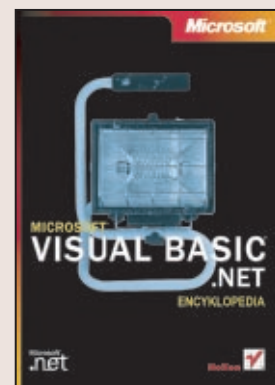
Autor: Microsoft Corporation

Tłumaczenie: Agata Bulandra

ISBN: 83-7197-822-7

Tytuł oryginału: [Microsoft Visual Basic .Net
Language Reference](#)

Format: B5, stron: 562



Wcześniejsze wersje języka Visual Basic tworzone były z myślą o projektowaniu aplikacji klienckich Microsoft® Windows®. Tworząc Visual Basic .NET, pomyślano także o zastosowaniu tego języka do tworzenia aplikacji internetowych i serwisów XML. Właśnie dlatego Visual Basic .NET generuje kod dla wspólnego środowiska uruchomieniowego, co spowodowało wprowadzenie zmian w obrębie samego języka.

Książka zawiera szczegółowe omówienie wszystkich elementów języka Visual Basic .NET opisanych w oficjalnej, elektronicznej dokumentacji. Jeżeli zajmujesz się programowaniem w VB .NET możesz być pewien, że szybko nie odłożysz jej na półkę.

Poznaj wszystkie szczegóły języka i środowiska uruchomieniowego:

- Atrybuty
- Stałe
- Typy danych
- Dyrektywy
- Funkcje
- Słowa kluczowe
- Metody
- Obiekty
- Operatory
- Właściwości
- Polecenia

Visual Basic .NET został zaprojektowany jako najprostsze, a przy tym efektywne narzędzie do tworzenia aplikacji i serwisów Microsoft .NET. Użyj Encyklopedii, a poznasz moc i elastyczność tego języka.



Spis treści

Zanim zaczniesz	13
Konwencje typograficzne i konwencje kodu	13
Rozdział 1. Visual Basic .NET — wprowadzenie	15
Zmiany w języku Visual Basic.....	15
Krótki przegląd zmian w implementacji elementów języka	15
Zmiany w deklaracjach	22
Zmiany w funkcjach.....	26
Zmiany w obiektach i komponentach	30
Zmiany w procedurach.....	35
Zmiany w przebiegu sterowania	39
Zmiany w tablicach	41
Zmiany w typach danych	44
Zmiany we właściwościach	47
Inne zmiany.....	50
Przegląd pojęć języka Visual Basic	52
Słowa kluczowe języka Visual Basic	58
Rozdział 2. Opis A – Z	61
#Const — dyrektywa.....	61
#ExternalSource — dyrektywa	62
#If..Then...#Else — dyrektywy	62
#Region — dyrektywa	64
& — operator	65
&= — operator	66
* — operator.....	66
*= — operator	67
+ — operator	68
+= — operator	71
— operator.....	72
-= — operator	73
/ — operator	74
/= — operator	75
= — operator	76
\ — operator	77
\= — operator	78

^ — operator	79
^= — operator	80
Add — metoda	81
AddHandler — polecenie	83
AddressOf — operator	84
Alias	85
And — operator	85
AndAlso — operator	87
Ansi	89
AppActivate — funkcja	89
AppWinStyle — wyliczenie	90
As	91
Asc, AscW — funkcje	91
Assembly	92
Auto	93
Beep — funkcja	93
Boolean — typ danych	93
ByRef	94
Byte — typ danych	94
ByVal	95
Call — polecenie	95
CallByName — funkcja	96
CallType — wyliczenie	98
Case	98
Char — typ danych	99
ChDir — funkcja	99
ChDrive — funkcja	100
Choose — funkcja	101
Chr, ChrW — funkcje	102
Class — polecenie	104
Clear — metoda	107
Collection — obiekt	108
ComClassAttribute — klasa	109
ComClassAttribute — konstruktor	110
ComClassAttribute.ClassID — właściwość	111
ComClassAttribute.EventID — właściwość	112
ComClassAttribute.InterfaceID — właściwość	112
ComClassAttribute.InterfaceShadows — właściwość	113
Command — funkcja	113
CompareMethod — wyliczenie	114
Const — polecenie	114
Count — właściwość	117
CreateObject — funkcja	118
CType — funkcja	120
CurDir — funkcja	121
Date — typ danych	122
DateAdd — funkcja	122
DateDiff — funkcja	125
DateFormat — wyliczenie	129
DateInterval — wyliczenie	129

DatePart — funkcja	130
DateSerial — funkcja	133
DateString — właściwość	135
DateValue — funkcja	136
Day — funkcja	137
DDB — funkcja	138
Decimal — typ danych	141
Declare — polecenie	141
Default	146
Delegate — polecenie	146
DeleteSetting — funkcja	150
Description — właściwość	152
Dim — polecenie	152
Dir — funkcja	159
DirectCast	160
Do...Loop — polecenia	161
Double — typ danych	162
DueDate — wyliczenie	163
Each	163
Else	164
Elseif	164
End	164
End — polecenie	167
Enum — polecenie	168
Environ — funkcja	171
EOF — funkcja	172
Erase — polecenie	174
Erl — właściwość	174
Err — obiekt	175
Error	176
Error — polecenie	176
ErrorToString — funkcja	178
Event — polecenie	179
Exit — polecenie	183
Explicit	185
False	185
FileAttr — funkcja	185
FileAttribute — wyliczenie	186
FileClose — funkcja	187
FileCopy — funkcja	188
FileDateTime — funkcja	189
FileGet — funkcja	189
FileGetObject — funkcja	194
FileLen — funkcja	198
FileOpen — funkcja	199
FilePut — funkcja	201
FilePutObject — funkcja	206
FileWidth — funkcja	210
Filter — funkcja	211
FirstDayOfWeek — wyliczenie	213

FirstWeekOfYear — wyliczenie	214
For	215
For Each...Next — polecenie	215
For...Next — polecenie	216
Format — funkcja	218
FormatCurrency — funkcja	220
FormatDateTime — funkcja	222
FormatNumber — funkcja	223
FormatPercent — funkcja	225
Formaty przeznaczone dla różnych wartości numerycznych (funkcja Format).....	226
FreeFile — funkcja.....	227
Friend	228
Function — polecenie	228
Funkcje konwersji typu	235
Funkcje matematyczne	240
Funkcje wywodzące się z funkcji matematycznych.....	243
FV — funkcja.....	244
Get — polecenie.....	246
GetAllSettings — funkcja	248
GetAttr — funkcja.....	249
GetChar — funkcja	251
GetEnumerator — metoda	252
GetException — funkcja	252
GetObject — funkcja	253
GetSetting — funkcja.....	256
GetType — operator.....	258
GoTo — polecenie	258
Handles.....	259
HelpContext — właściwość	261
HelpFile — właściwość	262
Hex — funkcja	263
Hour — funkcja.....	264
If...Then...Else — polecenia	264
IIf — funkcja	267
Implements	268
Implements — polecenie.....	268
Imports — polecenie	270
In.....	271
Inherits — polecenie	272
Input — funkcja	272
InputBox — funkcja.....	274
InputString — funkcja.....	276
InStr — funkcja	277
InStrRev — funkcja	278
Int, Fix — funkcje	280
Integer — typ danych	281
Interface — polecenie	282
IPmt — funkcja	285
IRR — funkcja	287
Is	289

Is — operator.....	289
IsArray — funkcja.....	290
IsDate — funkcja	291
IsDBNull — funkcja	292
IsError — funkcja.....	293
IsNothing — funkcja.....	294
IsNumeric — funkcja	295
IsReference — funkcja.....	296
Item — właściwość	297
Join — funkcja	298
Kill — funkcja.....	299
LastDLLError — właściwość	300
LBound — funkcja.....	301
LCase — funkcja.....	302
Left — funkcja	303
Len — funkcja.....	304
Lib	305
Like — operator	305
LineInput — funkcja	308
Loc — funkcja.....	309
Lock, Unlock — funkcje.....	310
LOF — funkcja	312
Long — typ danych.....	313
Loop	313
LSet — funkcja	314
LTrim, RTrim i Trim — funkcje.....	314
Me.....	315
Mid — funkcja	316
Mid — polecenie.....	317
Minute — funkcja	318
MIRR — funkcja.....	318
MkDir — funkcja	320
Mod — operator	321
Module	322
Module — polecenie	322
Month — funkcja	324
MonthName — funkcja.....	325
MsgBox — funkcja	326
MsgBoxResult — wyliczenie.....	329
MsgBoxStyle — wyliczenie.....	329
MustInherit	330
MustOverride	330
MyBase.....	331
MyClass.....	331
Namespace — polecenie	332
New	333
Next	334
Not — operator.....	334
Nothing.....	336
NotInheritable.....	336

NotOverridable	336
Now — właściwość	337
NPer — funkcja	338
NPV — funkcja	340
Number — właściwość	342
Obiekty	343
Object — typ danych	343
Oct — funkcja	343
Off	344
On	345
On Error — polecenie	345
OpenAccess — wyliczenie	349
OpenMode — wyliczenie	349
OpenShare — wyliczenie	350
Operator — krótki przegląd	350
Option	351
Option Compare — polecenie	351
Option Explicit — polecenie	352
Option Strict — polecenie	354
Optional	355
Or — operator	355
OrElse — operator	357
Overloads	358
Overridable	359
Overrides	359
ParamArray	359
Partition — funkcja	360
Pmt — funkcja	362
PPmt — funkcja	364
Predefiniowane formaty daty i czasu (Funkcja Format)	367
Predefiniowane formaty numeryczne (funkcja Format)	368
Preserve	369
Print, PrintLine — funkcje	369
Private	371
Property — polecenie	372
Protected	376
Public	377
PV — funkcja	377
QBColor — funkcja	379
Raise — metoda	380
RaiseEvent — polecenie	383
Randomize — polecenie	385
Rate — funkcja	386
ReadOnly	389
ReDim — polecenie	389
REM — polecenie	391
Remove — metoda	392
RemoveHandler — polecenie	393
Rename — funkcja	394
Replace — funkcja	395
Reset — funkcja	397

Resume	397
Resume — polecenie	398
Return — polecenie	399
RGB — funkcja	400
Right — funkcja	401
RmDir — funkcja	402
Rnd — funkcja	403
RSet — funkcja	405
SaveSetting — funkcja	405
ScriptEngine — właściwość	407
ScriptEngineBuildVersion — właściwość	407
ScriptEngineMajorVersion — właściwość	408
ScriptEngineMinorVersion — właściwość	409
Second — funkcja	410
Seek — funkcja	410
Select...Case — polecenie	413
Set — polecenie	415
SetAttr — funkcja	417
Shadows	418
Shared	419
Shell — funkcja	419
Short — typ danych	421
Single — typ danych	422
SLN — funkcja	422
Source — właściwość	424
Space — funkcja	425
SPC — funkcja	426
Split — funkcja	427
Stałe drukowania i wyświetlania	428
Static	429
Step	429
Stop — polecenie	429
Str — funkcja	430
StrComp — funkcja	431
StrConv — funkcja	433
StrDup — funkcja	435
String — typ danych	436
StrReverse — funkcja	437
Structure — polecenie	437
Sub — polecenie	440
Switch — funkcja	446
SYD — funkcja	447
SyncLock — polecenie	449
SystemTypeName — funkcja	450
TAB — funkcja	451
Then	452
Throw — polecenie	452
TimeOfDay — właściwość	453
Timer — właściwość	454
TimeSerial — funkcja	454
TimeString — właściwość	456

TimeValue — funkcja	457
To	458
Today — właściwość	458
Tristate — wyliczenie	459
True	460
Try...Catch...Finally — polecenia	460
TypeName — funkcja	462
typeof	464
Typy danych zdefiniowane przez użytkownika	464
UBound — funkcja	465
UCase — funkcja	466
Unicode	467
Until	467
Val — funkcja	467
VariantType — wyliczenie	469
VarType — funkcja	470
VBFixedArrayAttribute — klasa	471
VBFixedArrayAttribute — konstruktor	472
VBFixedArrayAttribute.FirstBound — pole	473
VBFixedArrayAttribute.SecondBound — pole	473
VBFixedStringAttribute — klasa	473
VBFixedStringAttribute — konstruktor	474
VBFixedStringAttribute.SizeConst — pole	475
VbStrConv — wyliczenie	475
VbTypeName — funkcja	476
Wartości zwracane przez funkcję CStr	477
Weekday — funkcja	477
WeekdayName — funkcja	479
When	480
While	481
While...End While — polecenia	481
With...End With — polecenia	482
WithEvents	483
Write, WriteLine — funkcje	484
WriteOnly	485
Xor — operator	486
Year — funkcja	488
Zdefiniowane przez użytkownika formaty daty i czasu (funkcja Format)	488
Zdefiniowane przez użytkownika formaty numeryczne (funkcja Format)	491

Dodatek A Operatory

495

Podział operatorów ze względu na funkcjonalność	495
Priorytet operatorów w języku Visual Basic	495
Operatory arytmetyczne	497
Operatory logiczne (bitowe)	497
Operatory porównania	498
Operatory przypisania	500
Operatory sklejania	501
Inne operatory	501

Dodatek B Podsumowania	503
Błędy — podsumowanie	503
Data i czas — podsumowanie	503
Deklaracje i stałe — podsumowanie	504
Dyrektywy kompilatora — podsumowanie	505
Funkcje finansowe — podsumowanie	505
Funkcje matematyczne — podsumowanie	506
Interakcje z otoczeniem — podsumowanie	506
Katalogi i pliki — podsumowanie	507
Konwersja — podsumowanie	507
Obiekt kolekcji — podsumowanie	508
Operacje na łańcuchach — podsumowanie	508
Przebieg sterowania — podsumowanie	509
Rejestr — podsumowanie	510
Tablice — podsumowanie	510
Typ danych — podsumowanie	511
Typy danych — podsumowanie	513
Wejście-wyjście — podsumowanie	513
Dodatek C Zestawienie składowych	515
Składowe biblioteki wykonawczej języka Visual Basic	515
Składowe obiektu Err	518
Składowe obiektu Collection	518
Składowe klasy ComClassAttribute	519
Składowe klasy VBFixedArrayAttribute	519
Składowe klasy VBFixedStringAttribute	520
Podział słów kluczowych i składowych według zadań	520
Dodatek D Funkcje konwersji	523
Dodatek E Opcje kompilatora języka Visual Basic	525
Opcje kompilatora języka Visual Basic w porządku alfabetycznym	525
Opcje kompilatora języka Visual Basic według funkcjonalności	527
@ (Określa plik odpowiedzi)	529
/addmodule	530
/baseaddress	531
/bugreport	532
/debug	533
/define	534
/delaysign	535
/help, /?	536
/imports	537
/keycontainer	537
/keyfile	538
/libpath	539
/linkresource	540
/main	541
/nologo	542
/nowarn	543
/optimize	543

/optioncompare	544
/optioncompare:binary	544
/optioncompare:text	545
/optionexplicit	546
/optionstrict	547
/out	547
/quiet	548
/recurse	549
/reference	550
/removeintchecks	551
/resource	552
/rootnamespace	553
/target	554
/target:exe	555
/target:library	556
/target:module	557
/target:winexe	557
/utf8output	559
/verbose	559
/warnaserror	560
/win32icon	561
/win32resource	561

2

Opis A – Z

#Const — dyrektywa

Definiuje stałe warunkowe dla kompilatora języka Visual Basic.

<code>#Const nazwaStałej = wyrażenie</code>

Części

nazwaStałej

Wymagana. Łącuch. Nazwa definiowanej stałej.

wyrażenie

Wymagana. Stała literowa, inna stała warunkowa kompilatora lub ich dowolna kombinacja zawierająca pewne lub wszystkie operatory arytmetyczne lub logiczne, za wyjątkiem **Is**.

Komentarze

Stałe warunkowe kompilatora zawsze są prywatne dla pliku, w którym istnieją. Za pomocą dyrektywy **#Const** nie można stworzyć publicznej zmiennej kompilatora; można ją stworzyć wyłącznie w interfejsie użytkownika.

W *wyrażeniu* można używać wyłącznie stałych warunkowych kompilatora lub stałych literowych. Zastosowanie standardowej stałej zdefiniowanej przez dyrektywę **#Const** spowoduje błąd. Odwrotnie — stałych zdefiniowanych przy użyciu słowa kluczowego **#Const** można używać tylko dla kompilacji warunkowej. Stałe mogą być także niezdefiniowane (mają wtedy wartość **Nothing**).

Przykład

Przykład wykorzystuje dyrektywę **#Const**.

```
#Const MojKraj = "Polska"  
#Const Wersja = "8.0.0012"  
#Const NrKlienta = 36
```

Zobacz także:

- ♦ Dyrektywy #If...Then...#Else;
- ♦ Polecenie Const.

#ExternalSource — dyrektywa

Wskazuje odwzorowanie pomiędzy liniami kodu źródłowego, a tekstem znajdującym się poza źródłem (na przykład w pliku .aspx).

```
#ExternalSource( StałaŁancuchowa , StałaLiczbowa )  
[ LogicznaLinia+ ]  
#End ExternalSource
```

Części

StałaŁancuchowa

Ścieżka do zewnętrznego źródła.

StałaLiczbowa

Numer pierwszej linii w zewnętrznym źródle.

LogicznaLinia

Linia, w której pojawia się błąd w źródle zewnętrznym.

#End ExternalSource

Kończy blok #ExternalSource.

Komentarze

Plik źródłowy może zawierać dyrektywy źródła zewnętrznego, które sygnalizują odwzorowanie między liniami źródłowymi, a tekstem na zewnątrz pliku źródłowego, więc błędy napotkane przy kompilacji identyfikowane są jako błędy pochodzące ze źródła zewnętrznego. Dyrektywy źródła zewnętrznego nie mają wpływu na kompilację i nie mogą być zagnieżdżane. Przeznaczone są wyłącznie do użytku wewnętrznego aplikacji.

#If...Then...#Else — dyrektywy

Warunkowo kompilują wybrane bloki kodu w języku Visual Basic.

```
#If wyrażenie Then  
instrukcje  
[ #ElseIf wyrażenie Then  
[ instrukcje ]
```

```
...  
#ElseIf wyrażenie Then  
  [ instrukcje ] ]  
[ #Else  
  [ instrukcje ] ]  
#End If
```

Części

wyrażenie

Wymagane jest dla instrukcji **If** i **ElseIf**, w pozostałych miejscach jest opcjonalne. Każde wyrażenie składa się z jednej lub kilku warunkowych stałych kompilatora, stałych literowych oraz operatorów, które daje w wyniku wartość **True** lub **False**. Istnieją trzy stałe warunkowej kompilacji: **Config**, **Debug** i **Trace**. **Debug** i **Trace** są typu **Boolean** i mogą być ustawione w oknie dialogowym właściwości projektu (*Project Properties*). Gdy zdefiniowana jest stała **Debug**, metody klasy **Debug** wyświetlają komunikaty w oknie **Output**. Gdy stała nie jest zdefiniowana, metody klasy **Debug** nie są kompilowane, także komunikaty dotyczące błędów nie są generowane. Podobnie dzieje się, gdy zdefiniowana zostaje stała **Trace** — metody klasy **Trace** wyświetlają dane wyjściowe w oknie **Output**. Gdy nie jest zdefiniowana, metody klasy **Trace** nie są kompilowane, a dane wyjściowe **Trace** nie są tworzone. Typ danych stałej **Config** to łańcuch, który odpowiada aktualnym ustawieniom w menedżerze konfiguracji (*Configuration Manager*).

instrukcje

Wymagane w bloku polecenia **If**, w innych miejscach są opcjonalne. Są to linie programu języka Visual Basic lub dyrektywy kompilatora, które są kompilowane, gdy skojarzone wyrażenia mają wartość **True**.

#End If

Kończy blok polecenia **#If**.

Komentarze

Ogólnie rzecz biorąc, działanie dyrektyw **#If...Then...#Else** jest takie samo jak działanie poleceń **If...Then...Else**. Jednak istnieje różnica — dyrektywy **#If...Then...#Else** określają to, co zostanie skompilowane przez kompilator, a polecenia **If...Then...Else** oceniają warunki podczas wykonania programu.

Kompilacja warunkowa zazwyczaj wykorzystywana jest do kompilowania tego samego programu dla różnych platform. Stosuje się ją także wtedy, gdy nie chce się umieszczać w pliku wykonywalnym kodu generującego dodatkowe informacje o błędach. Kod, wykluczony podczas kompilacji warunkowej, jest całkowicie pominięty w końcowym pliku wykonywalnym, a więc nie ma wpływu ani na jego wielkość, ani na wydajność.

Bez względu na rezultat wyliczeń, wszystkie wyrażenia wyliczane są przy użyciu polecenia **Option Compare Text**. Polecenie to (**Option Compare**) nie ma wpływu na wyrażenia znajdujące się w instrukcjach **#If** i **#ElseIf**.



Nie istnieją jednoliniowe wersje dyrektyw **#If**, **#Else**, **#ElseIf** i **#End If**. Oznacza to, że w tej samej linii — poza dyrektywami — nie może znajdować się żaden inny kod.

Przykład

W przykładzie wykorzystano konstrukcję **#If...Then...#Else**, aby ustalić, czy należy kompilować pewne instrukcje.

```
#Const NrKlienta = 36
#If NrKlienta = 35 Then
    ' Tu należy wstawić kod, który ma być skompilowany dla klienta nr 35.
#ElseIf NrKlienta = 36 Then
    ' Tu należy wstawić kod, który ma być skompilowany dla klienta nr 36.
#Else
    ' Tu należy wstawić kod, który ma być skompilowany dla pozostałych klientów.
#End If
```

Zobacz także:

- ♦ Dyrektywa **#Const**.

#Region — dyrektywa

Zwija i ukrywa fragmenty kodu w plikach Visual Basic .NET podczas edycji kodu w środowisku Visual Studio.

```
#Region "Łańcuch_identyfikatora"
#End Region
```

Części

Łańcuch_identyfikatora

Wymagana. Łańcuch stanowiący tytuł zwiniętego obszaru. Fragmenty te są domyślnie w stanie zwiniętym.

#End Region

Kończy blok **#Region**.

Komentarze

Dyrektywa **#Region** pozwala określić blok kodu, który można zwijać i rozwijać, gdy korzystamy z widoku konspektu w edytorze kodu w Visual Studio®. Instrukcja **#Region** obsługuje bloki semantyczne (na przykład **#If...#End If**). Oznacza to, że ich początek i koniec musi się znajdować wewnątrz tego samego bloku kodu.

Przykład

Poniższy przykład zawiera dyrektywę **#Region**.

```
#Region "FunkcjeMatematyczne"
    ' W tym miejscu wstawiamy cały kod związany z funkcjami matematycznymi.
#End Region
```

& — operator

Tworzy łańcuch będący połączeniem dwóch wyrażeń.

```
wynik = wyrażenie1 & wyrażenie2
```

Części

wynik

Wymagana. Dowolna zmienna typu **String** lub **Object**.

wyrażenie1

Wymagana. Dowolne wyrażenie.

wyrażenie2

Wymagana. Dowolne wyrażenie.

Komentarze

Jeśli typ danych *wyrażenia1* lub *wyrażenia2* jest inny niż **String**, to zostaną one przekształcone na łańcuch. Typ danych *wyniku* to **String**. Jeśli jedno z wyrażeń (lub obydwa) określone jest jako **Nothing** lub ma wartość **DBNull.value**, to traktowane jest jako łańcuch o wartości "".

Przykład

Operator **&** został wykorzystany do wymuszenia sklejania dwóch łańcuchów. Wynikiem jest wartość łańcucha, będąca połączeniem dwóch łańcuchów operandów.

```
Dim mojLancuch As String
mojLancuch = "Witam" & " Świat!" ' Zwraca "Witam świat!"
```

Poniższy przykład wykorzystuje operator **&**, aby wymusić konkatencję na wyniku wyszukiwania w bazie. Wynik jest łańcuchem uzyskanym z bazy lub — w przypadku, gdy wyszukiwanie zwróci wartość pustą — łańcuchem o zerowej długości.

```
Dim RS As Recordset = Cmd.Execute("Select * from ...")
Dim mojLancuch As String
MojLancuch = rs("au_id") & ""
```

Zobacz także:

- ◆ Operator **&=**;
- ◆ Operatory sklejania;
- ◆ Priorytet operatorów w języku Visual Basic;
- ◆ Podział operatorów ze względu na funkcjonalność.

&= — operator

Dokleja wyrażenie łańcuchowe do łańcucha zmiennej, przypisując zmiennej wynik.

`zmienna &= wyrażenie`

Części

zmienna

Wymagana. Dowolna zmienna typu **String**.

wyrażenie

Wymagana. Dowolne wyrażenie łańcuchowe.

Przykład

W poniższym przykładzie operator **&=** wykorzystano do połączenia dwóch zmiennych typu **String** i przypisania wyniku do pierwszej zmiennej.

```
Dim zmienna1 As String = "Witam "  
Dim zmienna2 As String = "świat!"  
zmienna1 &= zmienna2 ' Teraz wartością zmiennej zmienna1 jest "Witam świat!"
```

Zobacz także:

- ◆ Operator &;
- ◆ Operator * =;
- ◆ Operator + =;
- ◆ Operator — =;
- ◆ Operator / =;
- ◆ Operator =;
- ◆ Operator \ =;
- ◆ Operator ^ =.

* — operator

Tworzy iloczyn dwóch liczb.

`liczba1 * liczba2`

Części

liczba1

Wymagana. Dowolne wyrażenie numeryczne.

liczba2

Wymagana. Dowolne wyrażenie numeryczne.

Wynik

Wynik jest iloczynem *liczby1* i *liczby2*.

Obsługiwane typy

Byte, Short, Integer, Long, Single, Double, Decimal

Komentarze

Typ danych wyniku jest taki sam jak typ operandu o większym zakresie. Kolejność typów danych według skali od najmniejszego do największego zakresu jest następująca: Byte, Short, Integer, Long, Single, Double i Decimal.

Jeśli wartość wyrażenia jest podana jako **Nothing** lub jest pusta, to traktowana jest jako 0.

Przykład

Poniższy przykład wykorzystuje operator * do przemnożenia dwóch liczb. Wynik jest iloczynem dwóch operandów.

```
Dim mojaWartosc As Double
mojaWartosc = 2 * 2           ' Zwraca 4.
mojaWartosc = 459.35 * 334.90 ' Zwraca 153836.315.
```

Zobacz także:

- ◆ Operator * =;
- ◆ Operatory arytmetyczne;
- ◆ Priorytet operatorów w języku Visual Basic;
- ◆ Podział operatorów ze względu na funkcjonalność.

* = — operator

Mnoży wartość zmiennej przez wartość wyrażenia i przypisuje wynik zmiennej.

zmienna *= *wyrażenie*

Części

zmienna

Wymagana. Dowolna zmienna numeryczna.

wyrażenie

Wymagana. Dowolne wyrażenie numeryczne.

Przykład

Poniższy przykład wykorzystuje operator `*=` do pomnożenia zmiennej, będącej liczbą całkowitą typu **Integer**, przez drugą zmienną i przypisuje wynik do pierwszej zmiennej.

```
Dim zmienna1 As Integer = 10
Dim zmienna2 As Integer = 3
zmienna1 *= zmienna2 ' Wartość zmiennej zmienna1 wynosi teraz 30.
```

Zobacz także:

- ◆ Operator `&=`;
- ◆ Operator `*`;
- ◆ Operator `+=`;
- ◆ Operator `—=`;
- ◆ Operator `/=`;
- ◆ Operator `=`;
- ◆ Operator `\=`;
- ◆ Operator `^=`.

+ — operator

Dodaje dwie liczby. Jest także wykorzystywany do łączenia dwóch łańcuchów.

<code>wyrażenie1 + wyrażenie2</code>

Części

wyrażenie1

Wymagana. Dowolne wyrażenie numeryczne lub łańcuch.

wyrażenie2

Wymagana. Dowolne wyrażenie numeryczne lub łańcuch.

Wynik

W przypadku, gdy *wyrażenie1* i *wyrażenie2* są wyrażeniami numerycznymi, wartością wyniku będzie suma wyrażeń *wyrażenie1* i *wyrażenie2*. Jeśli *wyrażenie1* i *wyrażenie2* są łańcuchami, to wartość wyniku będzie sklejaniem łańcuchów *wyrażenie1* i *wyrażenie2*.

Obsługiwane typy

Byte, Short, Integer, Long, Single, Double, Decimal, String

Komentarze

Jeśli korzystamy z operatora +, może okazać się, że nie będziemy w stanie ustalić, czy zajdzie dodawanie, czy sklejanie łańcuchów. W celu uniknięcia niejasności, a także po to, by otrzymać łatwy do zrozumienia kod, należy do łączenia łańcuchów używać operatora &.

Jeśli żadne z wyrażeń nie jest typu **Object**, stosowane są przedstawione niżej zasady.

Jeśli...	Wtedy...
Obydwa wyrażenia mają ten sam typ danych numerycznych (Byte , Short , Integer , Long , Single , Double lub Decimal)	Zostaną dodane.
Obydwa wyrażenia są łańcuchami	Zostaną sklejone.
Jedno wyrażenie jest typu numerycznego, a drugie jest łańcuchem	Jeśli Option Strict ma wartość On , zostanie wygenerowany błąd kompilacji; jeśli Option Strict jest wyłączony (Off), łańcuch będzie niejawnie konwertowany na typ Double i wyrażenia zostaną dodane. Jeśli łańcuch nie będzie mógł zostać przekształcony na wartość numeryczną, zgłoszony zostanie wyjątek InvalidCastException .
Jedno wyrażenie jest typu numerycznego, a drugie ma wartość Nothing	Jeśli Option Strict ma wartość On , to zostanie wygenerowany błąd kompilacji; jeśli Option Strict jest wyłączony (Off), wyrażenia zostaną dodane (wartość Nothing zostanie potraktowana jako zero).
Jedno wyrażenie jest łańcuchem, a drugie ma wartość Nothing	Jeśli Option Strict ma wartość On , zostanie wygenerowany błąd kompilacji; jeśli Option Strict jest wyłączony, wyrażenia zostaną sklejone, a wartość Nothing zostanie zamieniona na wartość "".

Jeśli jedno z wyrażeń jest typu **Object**, stosowane są przedstawione niżej zasady.

Jeśli...	Wtedy...
Jedno wyrażenie jest wyrażeniem numerycznym typu Object , a drugie ma wartość numeryczną	Jeśli Option Strict jest włączone (On), zostanie wygenerowany błąd kompilacji; jeśli Option Strict jest wyłączony, wyrażenia zostaną dodane.
Jedno wyrażenie jest wyrażeniem numerycznym typu Object , a typ drugiego to String	Jeśli Option Strict ma wartość On , zostanie wygenerowany błąd kompilacji; jeśli Option Strict jest wyłączony (Off), łańcuch będzie niejawnie konwertowany na typ Double i wyrażenia zostaną dodane. Jeśli łańcuch nie będzie mógł zostać przekształcony na wartość numeryczną, zgłoszony zostanie wyjątek InvalidCastException .
Jedno wyrażenie jest łańcuchem typu Object , a typ drugiego to String	Jeśli Option Strict ma wartość On , zostanie wygenerowany błąd kompilatora; jeśli Option Strict jest wyłączony (Off), typ Object będzie niejawnie konwertowany na String , a potem wyrażenia zostaną połączone.

Jeśli...	Wtedy...
Jedno wyrażenie jest łańcuchem typu Object , a drugie wyrażenie ma wartość numeryczną	Jeśli Option Strict ma wartość On , zostanie wygenerowany błąd kompilatora; jeśli Option Strict jest wyłączone (Off), łańcuch będzie niejawnie konwertowany na typ Double i wyrażenia zostaną dodane. Jeśli łańcuch nie będzie mógł zostać przekształcony na wartość numeryczną, zgłoszony zostanie wyjątek InvalidCastException .

Jeśli obydwa wyrażenia są typu **Object**, stosowane są przedstawione niżej zasady (jedynie w przypadku **Option Strict Off**).

Jeśli...	Wtedy...
Obydwa wyrażenia typu Object są numeryczne	Zostają dodane.
Obydwa wyrażenia typu Object są łańcuchami	Zostają sklejone.
Jedno wyrażenie typu Object jest numeryczne, a drugie jest łańcuchem	Niejawnie zmieniony zostaje nienumeryczny Object na Double i dodany. Jeśli Object nie może zostać przekształcony na wartość numeryczną, zgłoszony zostanie wyjątek InvalidCastException .

Jeśli jedno lub obydwa wyrażenia będą opisane jako **Nothing** lub będą miały przypisaną wartość **DBNull**, traktowane będą jako łańcuch o wartości "".

Przykład

Przykład wykorzystuje operator + do dodania liczb. Operatora + można także używać do łączenia łańcuchów. Aby uniknąć niejasności, powinno się używać w takim przypadku operatora &. Jeśli komponenty wyrażenia utworzonego za pomocą operatora + są numeryczne, to wykonana zostanie operacja arytmetyczna. Jeśli komponenty są jawnymi łańcuchami, to zostaną sklejone. Wyrażenie nie może składać się z komponentów różnych typów. Wynik operacji arytmetycznej zwraca sumę dwóch operandów. Wynik sklejania zwraca łańcuch, który jest połączeniem dwóch operandów.

```
Dim mojaLiczba As Integer
Dim zmienna1 As String
Dim zmienna2 As Integer
mojaLiczba = 2 + 2           ' Zwraca 4.
mojaLiczba = 4257.04 + 98112 ' Zwraca 102369.04.
```

```
Option Strict On
' Inicjuje dwie zmienne różnych typów.
zmienna1 = "34"
zmienna2 = 6
mojaLiczba = zmienna1 + zmienna2 ' Generuje błąd podczas kompilacji.
```

```
Option Strict Off
zmienna1 = "34"
zmienna2 = 6
mojaLiczba = zmienna1 + zmienna2
```

' Zwraca 40 (sumę). Łańcuch ze zmiennej `zmienna1` został przekształcony na wartość numeryczną. Nie jest polecanym rozwiązaniem użycie `Option Strict Off` aby umożliwić przeprowadzanie takich operacji.

Zobacz także:

- ◆ Operator `&`;
- ◆ Operatory sklejania;
- ◆ Operatory arytmetyczne;
- ◆ Podział operatorów ze względu na funkcjonalność;
- ◆ Priorytet operatorów w języku Visual Basic.

+= — operator

Dodaje wartość wyrażenia do wartości zmiennej, a wynik przypisuje tej zmiennej. Dkleja także wyrażenie w postaci łańcucha do zmiennej typu **String** i przypisuje wynik do zmiennej.

```
zmienna += wyrażenie
```

Części

zmienna

Wymagana. Dowolna zmienna numeryczna lub łańcuch.

wyrażenie

Wymagana. Dowolne wyrażenie numeryczne lub łańcuch.

Komentarze

Jeśli środowisko kompilacji wymusza semantykę restrykcyjną, to taka instrukcja niejawnie wykona rozszerzenie (a nie zwężenie) konwersji. Jeżeli dozwolona jest semantyka liberalna, operator spowoduje wykonanie wielu różnych konwersji danych numerycznych i łańcuchowych. Przekształcenia są takie same jak przekształcenia dokonywane przez operator `+`. Więcej informacji na temat dokonywanych konwersji można znaleźć w podrozdziale — Operator `+`. Chcąc bliżej poznać zagadnienie semantyki liberalnej oraz restrykcyjnej, należy przeczytać podrozdział — Polecenie `Option Strict`.

Przykład

Poniższe przykłady używają operatora `+=`, aby połączyć wartość jednej zmiennej z wartością drugiej zmiennej. W pierwszym przykładzie operator `+=` użyty jest wraz ze zmiennymi numerycznymi w celu dodania ich do siebie. W drugim przykładzie operator wykorzystano do sklejania wartości dwóch łańcuchów. W obu przypadkach wynik przypisywany jest do pierwszej zmiennej.

```
Dim zmienna1 As Integer = 10
Dim zmienna2 As Integer = 3
zmienna1 += zmienna2 ' Wartość zmiennej zmienna1 wynosi teraz 13.
```

```
' Ten przykład wykorzystuje zmienne w postaci łańcuchów.  
Dim zmienna1 As String = "10"  
Dim zmienna2 As String = "3"  
zmienna1 += zmienna2 ' Wartość zmiennej zmienna1 wynosi teraz "103".
```

Zobacz także:

- ◆ Operator +;
- ◆ Operatory arytmetyczne;
- ◆ Operatory sklejania;
- ◆ Priorytet operatorów w języku Visual Basic;
- ◆ Podział operatorów ze względu na funkcjonalność.

— operator

Oblicza różnicę dwóch liczb lub sygnalizuje wartość ujemną wyrażenia numerycznego.

I. składnia

liczba1 - liczba2

II. składnia

-liczba

Części

liczba

Wymagana. Dowolne wyrażenie numeryczne.

liczba1

Wymagana. Dowolne wyrażenie numeryczne.

liczba2

Wymagana. Dowolne wyrażenie numeryczne.

Wynik

Wynikiem będzie różnica pomiędzy wyrażeniami *liczba1* i *liczba2*.

Obsługiwane typy

Byte, Short, Integer, Long, Single, Double, Decimal

Komentarze

W pierwszej składni operator – jest arytmetycznym operatorem odejmowania, używanym w celu obliczenia różnicy dwóch liczb. W drugiej składni operator – jest jednoargumentowym operatorem zmiany znaku wartości wyrażenia. Typ danych wyniku jest taki sam jak typ danych operandu o większym zakresie. Kolejność zakresów od najmniejszych do największych jest następująca: Byte, Short, Integer, Long, Single, Double i Decimal.

Jeśli wyrażenie ma wartość **Nothing**, to jego wartość traktowana jest jako zero.

Przykład

Poniższy przykład wykorzystuje operator – do obliczenia i zwrócenia różnicy dwóch liczb.

```
Dim mojWynik As Double
mojWynik = 4 - 2 ' Zwraca 2.
mojWynik = 459.35 - 334.9 ' Zwraca 124.45.
```

Zobacz także:

- ◆ Operator ==;
- ◆ Operatory arytmetyczne;
- ◆ Priorytet operatorów w języku Visual Basic;
- ◆ Podział operatorów ze względu na funkcjonalność.

== — operator

Odejmuje wartość wyrażenia od wartości zmiennej, a wynik przypisuje zmiennej.

zmienna -= wyrażenie

Części

zmienna

Wymagana. Dowolna zmienna numeryczna.

wyrażenie

Wymagana. Dowolne wyrażenie numeryczne.

Przykład

Poniższy przykład stosuje operator == w celu odjęcia zmiennej, która jest liczbą całkowitą, od innej zmiennej i przypisania wyniku do pierwszej zmiennej.

```
Dim zmienna1 As Integer = 10
Dim zmienna2 As Integer = 3
zmienna1 -= zmienna2 ' Wartość zmiennej zmienna1 wynosi teraz 7.
```

Zobacz także:

- ◆ Operator `&=`;
- ◆ Operator `*=`;
- ◆ Operator `+=`;
- ◆ Operator `-=`;
- ◆ Operator `/=`;
- ◆ Operator `=`;
- ◆ Operator `\=`;
- ◆ Operator `^=`.

/ — operator

Oblicza iloraz dwóch liczb i zwraca wynik jako liczbę zmiennoprzecinkową.

<code>liczba1 / liczba2</code>

Części

wynik

Wymagana. Dowolna zmienna numeryczna.

liczba1

Wymagana. Dowolne wyrażenie numeryczne.

liczba2

Wymagana. Dowolne wyrażenie numeryczne.

Wynik

Wynik jest ilorazem wartości wyrażenia *liczba1* przez wartość wyrażenia *liczba2*.

Obsługiwane typy

Byte, Short, Integer, Long, Single, Double, Decimal

Komentarze

Wynik jest zazwyczaj typu **Double**. W tabeli na następnej stronie wymienione zostały wyjątki od tej reguły.

Jeśli wyrażenie ma wartość **Nothing** lub jest puste, to jego wartość traktowana jest jako zero.

Jeśli...	Wtedy typ wyniku to...
Jedno wyrażenie ma typ Single , a typ drugiego jest inny niż Double	Single .
Obydwa wyrażenia mają typ Decimal	Decimal . Jeśli wyrażenie typu Decimal zostanie podzielone przez 0, zostanie zgłoszony wyjątek DividedbyZero . Wyjątek pojawia się jedynie przy wyrażeniach typu Decimal .

Przykład

Przykład wykorzystuje operator / przy wykonywaniu dzielenia zmiennopozycyjnego. Wynik jest ilorazem dwóch operandów.

```
Dim MojaWartosc As Double
MojaWartosc = 10 / 4 ' Zwraca 2.5.
MojaWartosc = 10 / 3 ' Zwraca 3.333333.
```

Zobacz także:

- ◆ Operator \=;
- ◆ Operator \;
- ◆ Operatory arytmetyczne;
- ◆ Priorytet operatorów w języku Visual Basic;
- ◆ Podział operatorów ze względu na funkcjonalność.

/= — operator

Dzieli wartość zmiennej przez wartość wyrażenia, a wynik przypisuje zmiennej.

zmienna /= wyrażenie

Części

zmienna

Wymagana. Dowolna zmienna numeryczna.

wyrażenie

Wymagana. Dowolne wyrażenie numeryczne.

Komentarze

Instrukcja ta przypisuje typ **Double** zmiennej znajdującej się po lewej stronie operatora. Jeśli polecenie **Option Strict** ma wartość **On**, zmienna musi przyjąć typ **Double**. Jeśli polecenie **Option Strict** jest wyłączone (**Off**), zostanie wykonana niejawna konwersja, a wartość wynikowa będzie przypisana do zmiennej, ale możliwe jest wystąpienie błędu w czasie wykonania programu. Więcej informacji można znaleźć w podrozdziale — Polecenie Option Strict.

Przykład

W poniższym przykładzie operator /= został wykorzystany do podzielenia jednej zmiennej, która jest liczbą całkowitą, przez inną zmienną, a także do przypisania wyniku do pierwszej zmiennej.

```
Dim zmienna1 As Integer = 12
Dim zmienna2 As Integer = 3
zmienna1 /= zmienna2 ' Wartość zmiennej zmienna1 wynosi 4.
```

Zobacz także:

- ◆ Operator &=;
- ◆ Operator * =;
- ◆ Operator +=;
- ◆ Operator -=;
- ◆ Operator /;
- ◆ Operator =;
- ◆ Operator \=;
- ◆ Operator ^=.

= — operator

Wykorzystywany do przypisywania wartości do zmiennej lub do właściwości.

<i>zmienna = wartość</i>

Części

zmienna

Dowolna zmienna lub dowolna zapisywalna właściwość.

wartość

Dowolny literał, stała lub wyrażenie.

Komentarze

Nazwa po lewej stronie znaku równości może być zwykłą zmienną skalarną, właściwością lub elementem tablicy. Po lewej stronie znaku równości mogą się znaleźć wyłącznie właściwości zapisywalne w czasie wykonywania programu. Wartość po prawej stronie znaku równości jest przypisywana zmiennej znajdującej się po lewej stronie znaku równości.

Przykład

Przykład prezentuje użycie operatora przypisania. Wartość po prawej stronie wyrażenia przypisywana jest do zmiennej znajdującej się po lewej stronie wyrażenia.

```
Dim mojaLiczba as Integer
Dim mojLancuch as String
Dim mojPrzycisk as System.Windows.Forms.Button
Dim mojObiekt as Object
mojaLiczba = 42
mojLancuch = "Oto przykład stałej łańcuchowej"
mojPrzycisk = New System.Windows.Forms.Button()
mojObiekt = mojaLiczba
mojObiekt = mojLancuch
mojObiekt = mojPrzycisk
```

Zobacz także:

- ◆ Operator &=;
- ◆ Operator *=;
- ◆ Operator +=;
- ◆ Operator -=;
- ◆ Operator /=;
- ◆ Operator \=;
- ◆ Operator ^=.

\ — operator

Dzieli dwie liczby przez siebie i zwraca wynik będący liczbą całkowitą.

<i>liczba1 \ liczba2</i>

Części

liczba1

Wymagana. Dowolne wyrażenie numeryczne będące liczbą całkowitą.

liczba2

Wymagana. Dowolne wyrażenie numeryczne będące liczbą całkowitą.

Obsługiwane typy

Byte, Short, Integer lub Long

Wynik

Wynik jest ilorazem wartości wyrażenia *liczba1* przez wartość wyrażenia *liczba2* z pominięciem reszty z dzielenia.

Komentarze

Jeśli poleceniu **Option Strict** przypisana zostanie wartość **Off**, wszelkie wyrażenia numeryczne zmiennoprzecinkowe będą konwertowane na wyrażenia typów **Byte**, **Short**, **Integer** lub **Long**, zanim zostanie wykonane dzielenie. Jeśli poleceniu **Option Strict** przypisana zostanie wartość **On**, to w czasie kompilacji zgłoszony zostanie błąd.

Typem danych wyniku będzie: **Byte**, **Short**, **Integer** lub **Long**. Dowolna reszta z dzielenia zostanie pominięta.

Jeśli któreś z wyrażeń zostanie określone jako **Nothing** lub **Empty**, to traktowane będzie jako zero. Próba wykonania dzielenia przez zero doprowadza do zgłoszenia wyjątku **DividedByZeroException**.

Przykład

W przykładzie skorzystano z operatora \ do wykonania dzielenia. Wynik jest liczbą całkowitą, przedstawiającą iloraz dwóch operandów.

```
Dim mojaWartosc As Integer
mojaWartosc = 11 \ 4 ' Zwraca 2.
mojaWartosc = 9 \ 3 ' Zwraca 3.
mojaWartosc = 100 \ 3 ' Zwraca 33.
mojaWartosc = 67 \ -3 ' Zwraca -22.
```

Zobacz także:

- ◆ Operator \=;
- ◆ Operator /;
- ◆ Operatory arytmetyczne;
- ◆ Priorytet operatorów w języku Visual Basic;
- ◆ Podział operatorów ze względu na funkcjonalność.

\= — operator

Dzieli wartość zmiennej przez wartość wyrażenia, a skrócony do liczby całkowitej wynik przypisuje do zmiennej.

<i>zmienna \= wyrażenie</i>

Części

zmienna

Wymagana. Dowolna zmienna numeryczna.

wyrażenie

Wymagana. Dowolne wyrażenie numeryczne.

Komentarze

Więcej informacji na temat dzielenia można znaleźć w podrozdziale — Operator \.

Przykład

W poniższym przykładzie operator \= posłużył do podzielenia zmiennej o wartości całkowitej przez drugą zmienną i przypisania do pierwszej zmiennej wyniku w postaci liczby całkowitej.

```
Dim zmienna1 As Integer = 10
Dim zmienna2 As Integer = 3
zmienna1 \= zmienna2 ' Wartość zmiennej zmienna1 wynosi teraz 3.
```

Zobacz także:

- ◆ Operator &=;
- ◆ Operator *=;
- ◆ Operator +=;
- ◆ Operator -=;
- ◆ Operator /=;
- ◆ Operator =;
- ◆ Operator \;
- ◆ Operator ^=.

^ — operator

Podnosi liczbę do potęgi wyrażonej drugą liczbą.

liczba ^ *wykładnik*

Części

liczba

Wymagana. Dowolne wyrażenie numeryczne.

wykładnik

Wymagana. Dowolne wyrażenie numeryczne.

Wynik

Wartość wyniku jest wartość wyrażenia *liczba* podniesioną do potęgi wyrażoną wartością wyrażenia *wykładnik*.

Obsługiwane typy

Double. Wszystkie operandy innych typów zostaną przekształcone na typ **Double**.

Komentarze

Liczba może być liczbą ujemną tylko wtedy, gdy *wykładnik* jest liczbą całkowitą. Jeśli w jednym wyrażeniu wykonywanych jest więcej podniesień do potęgi, operator ^ jest brany do obliczeń w kolejności wystąpienia od lewej do prawej.

Otrzymywany *Wynik* jest typu **Double**.

Przykład

W przykładzie zastosowano operator ^ do podniesienia liczby do potęgi wykładnika. Wynik to wartość pierwszego operandu podniesiona do potęgi o wykładniku równym wartości drugiego operandu.

```
Dim mojaWartosc As Double
mojaWartosc = 2 ^ 2      ' Zwraca 4.
mojaWartosc = 3 ^ 3 ^ 3  ' Zwraca 19683.
mojaWartosc = (-5) ^ 3    ' Zwraca -125.
mojaWartosc = (-5) ^ 4    ' Zwraca 625.
```

Zobacz także:

- ♦ Operator ^=;
- ♦ Operatory arytmetyczne;
- ♦ Priorytet operatorów w języku Visual Basic;
- ♦ Podział operatorów ze względu na funkcjonalność.

^ = — operator

Podnosi wartość zmiennej do potęgi o wykładniku równym wartości wyrażenia, a wynik przypisuje zmiennej.

<i>zmienna</i> ^= <i>wyrażenie</i>

Części

zmienna

Wymagana. Dowolna zmienna numeryczna.

wyrażenie

Wymagana. Dowolne wyrażenie numeryczne.

Przykład

W poniższym przykładzie wykorzystano operator `^=` do podniesienia wartości zmiennej, która jest liczbą typu **Integer**, do potęgi, która jest wartością drugiej zmiennej, i przypisania wyniku do pierwszej zmiennej.

```
Dim zmienna1 As Integer = 10
Dim zmienna2 As Integer = 3
zmienna1 ^= zmienna2 ' Wartość zmiennej zmienna1 wynosi teraz 1000.
```

Zobacz także:

- ◆ Operator `&=`;
- ◆ Operator `*=`;
- ◆ Operator `+=`;
- ◆ Operator `-=`;
- ◆ Operator `/=`;
- ◆ Operator `=`;
- ◆ Operator `\=`;
- ◆ Operator `^`.

Add — metoda

Dodaje elementy do obiektu kolekcji (**Collection**).

```
Public Sub Add(  
    ByVal Element As Object, _  
    Optional ByVal Klucz As String, _  
    Optional ByVal Przed As Object = Nothing, _  
    Optional ByVal Za As Object = Nothing  
)
```

Parametry

Element

Wymagany. Obiekt dowolnego typu, określający składową, która ma być dodana do kolekcji.

Klucz

Opcjonalny. Unikatowe wyrażenie łańcuchowe, określające łańcuch klucza, który może zostać użyty zamiast indeksu pozycyjnego w celu uzyskania dostępu do elementu kolekcji.

Przed

Opcjonalny. Wyrażenie określające położenie względne w kolekcji. Element, który ma być dodany, zostanie umieszczony w kolekcji przed elementem zidentyfikowanym przez argument *Przed*. Jeśli parametr *Przed* jest wyrażeniem

numerycznym, to jego wartość musi zawierać się w zakresie od wartości jeden do wartości właściwości **Count** danej kolekcji. Jeśli parametr ma postać łańcucha, to musi odpowiadać łańcuchowi klucza elementu kolekcji, do którego się odwołujemy (podanemu, gdy element był dodawany do kolekcji). Użycie parametrów *Przed* i *Za* wzajemnie się wyklucza.

Za

Opcjonalny. Wyrażenie określające położenie względne w kolekcji. Element, który ma być dodany, zostanie umieszczony w kolekcji za elementem zidentyfikowanym przez parametr *Za*. Jeśli parametr *Za* jest wyrażeniem numerycznym, to jego wartość musi zawierać się w zakresie od wartości jeden do wartości właściwości **Count** danej kolekcji. Jeśli parametr ma postać łańcucha, to musi odpowiadać łańcuchowi klucza elementu kolekcji, do którego się odwołujemy (podanemu, gdy element był dodawany do kolekcji). Użycie parametrów *Przed* i *Za* wzajemnie się wyklucza.

Wyjątki i błędy

Typ wyjątku	Numer błędu	Warunek
ArgumentException	5	Określone są jednocześnie obydwa parametry <i>Przed</i> i <i>Za</i> lub argument nie odnosi się do istniejącego elementu kolekcji.
ArgumentException	5	Podany <i>Klucz</i> już istnieje.

Komentarze

Argument *Przed* lub *Za* musi odwoływać się do istniejącego elementu kolekcji, w przeciwnym razie zostanie wywołany błąd.

Błąd zdarzy się także, gdy podana wartość *Klucz* będzie identyczna do klucza już istniejącego elementu kolekcji.

Przykład

W przykładzie wykorzystano metodę **Add**, aby dodać obiekty *Dziecko* — egzemplarze klasy o nazwie *Dziecko*, która zawiera publiczną właściwość *Imię* — do kolekcji o nazwie *Rodzina*. Aby sprawdzić, jak to działa, możemy stworzyć formularz z dwoma przyciskami. Jako wartość właściwości **Text** tych przycisków podajmy słowa „Dodaj” i „Wyświetl”. W kodzie formularza dodajmy definicję klasy **Dziecko** i deklarację kolekcji **Rodzina**. Teraz możemy zmienić zdarzenie **Click** przycisków **Dodaj** i **Wyświetl**, tak jak pokazano to w poniższym kodzie. Przycisk **Dodaj** umożliwia dodanie dzieci, a przycisk **Wyświetl** wyświetli imiona wszystkich dzieci.

```
Public Class Dziecko
    Public Imie As String
    Sub New(ByVal noweImie As String)
        Imie = noweImie
    End Sub
End Class
```

```

Private Rodzina As New Collection() ' Tworzymy obiekt kolekcji.

Private Sub Dodaj_Click(ByVal nadawca As System.Object, _
    ByVal e As System.EventArgs) Handles Button1.Click
    Dim noweImie As String
    noweImie = InputBox("Imię nowego członka rodziny: ")
    If noweImie <> "" Then
        Rodzina.Add(New Dziecko(noweImie), noweImie)
    End If
End Sub

Private Sub Wyświetl_Click(ByVal nadawca As System.Object, _
    ByVal e As System.EventArgs) Handles Button2.Click
    Dim dzieckoA As Dziecko
    For Each dzieckoA In Rodzina
        MsgBox(dzieckoA.Imie)
    Next
End Sub

```

Zobacz także:

- ◆ Właściwość Item;
- ◆ Metoda Remove;
- ◆ ArgumentException.

Stosowana do

Obiekt kolekcji.

AddHandler — polecenie

Kojarzy zdarzenie z procedurą obsługi zdarzeń.

AddHandler *zdarzenie*, AddressOf *procedura obsługi zdarzenia*

Części

zdarzenie

Nazwa zdarzenia, które należy obsłużyć.

procedura obsługi zdarzenia

Nazwa procedury, która obsłuży zdarzenie.

Komentarze

Polecenia **AddHandler** i **RemoveHandler** umożliwiają rozpoczęcie i zakończenie obsługi zdarzeń w dowolnym momencie w trakcie wykonywania programu.

Przykład

```

Sub testZdarzen()
    Dim Obj As New Klasa1()
    ' Kojarzy procedurę obsługi zdarzeń ze zdarzeniem.
    AddHandler Obj.Zdarzenie, AddressOf ProceduraObslugiZdarzenia
    Obj.SpowodujZdarzenie() ' Powodujemy, że obiekt wywoła zdarzenie.
End Sub

Sub ProceduraObslugiZdarzenia()
    ' Ta procedura obsługuje zdarzenia wywołane przez obiekt Obj.
    MsgBox("ProceduraObslugiZdarzenia wyłapała zdarzenie.") ' Obsługuje zdarzenie.
End Sub

Public Class Klasa1
    Public Event Zdarzenie() ' Deklaracja zdarzenia.
    Sub SpowodujZdarzenie()
        RaiseEvent Zdarzenie() ' Wywołanie zdarzenia.
    End Sub
End Class

```

Zobacz także:

- ◆ Polecenie RemoveHandler;
- ◆ Handles.

AddressOf — operator

Tworzy egzemplarz delegacji procedury, odwołujący się do określonej procedury.

AddressOf *nazwaProcedury*

Wymagana *nazwaProcedury* określa procedurę, do której będzie się odwoływać nowo utworzona delegacja procedury.

Komentarze

Operator **AddressOf** tworzy delegację funkcji, wskazując funkcję określoną przez *nazwęProcedury*. Gdy określona procedura jest metodą egzemplarza jakiegoś obiektu, to delegacja funkcji odnosi się zarówno do egzemplarza obiektu, jak i do metody. W związku z tym, gdy wywoływana jest delegacja funkcji, wywoływana jest także określona metoda określonego egzemplarza obiektu.

Operator **AddressOf** może być wykorzystywany jako operand konstruktora delegacji, a także może być użyty w kontekście, w którym typ delegacji może być ustalony przez kompilator.

Przykład

W poniższym przykładzie wykorzystano operator **AddressOf**, aby wyznaczyć delegację do obsługi zdarzenia **Click** przycisku.

```
Public Sub ObslugaKliknieciaPrzycisku(ByVal nadawca As Object, e As _
                                   System.EventArgs)
    ' Pominięto kod implementacji.
End Sub

Public Sub New()
    AddHandler Przycisk1.Click, AddressOf ObslugaKliknieciaPrzycisku
    ' Pominięto dodatkowy kod.
End Sub
```

Operator **AddressOf** został użyty w poniższym przykładzie do wyznaczenia funkcji początkowej wątku.

```
Public Sub LiczOwce()
    Dim i As Integer = 1 ' Owiec nie liczy się od 0.
    Do While (True) ' Wieczna pętla.
        Console.WriteLine("Owca " & i & " Beee")
        i = i + 1
        Thread.Sleep(1000) ' Czekamy 1 sekundę
    Loop
End Sub

Sub UzyjWatku()
    Dim t As New System.Threading.Thread(AddressOf LiczOwce)
    t.Start()
End Sub
```

Zobacz także:

- ◆ Polecenie Declare;
- ◆ Polecenie Function;
- ◆ Polecenie Sub.

Alias

Słowo kluczowe **Alias** sygnalizuje, że zewnętrzna procedura w bibliotece DLL posiada inną nazwę.

Słowo kluczowe **Alias** używane jest w kontekście:

Polecenie Declare.

Zobacz także:

- ◆ Słowa kluczowe języka Visual Basic.

And — operator

Oblicza iloczyn logiczny dwóch wyrażeń typu **Boolean** lub iloczyn bitowy dwóch wyrażeń numerycznych.

wynik = wyrażenie1 And wyrażenie2

Części

wynik

Wymagana. Dowolne wyrażenie logiczne lub numeryczne. Wynik działania logicznego jest wynikiem, będącym iloczynem logicznym dwóch wyrażeń. Wynik działania numerycznego jest wartością numeryczną, wynikającą z iloczynu logicznego na poziomie bitowym dwóch wyrażeń numerycznych.

wyrażenie1

Wymagana. Dowolne wyrażenie logiczne lub numeryczne.

wyrażenie2

Wymagana. Dowolne wyrażenie logiczne lub numeryczne.

Komentarze

Jeśli w porównaniu logicznym (danych typu **Boolean**) zarówno *wyrażenie1*, jak i *wyrażenie2* przyjmuje wartość **True**, to *wynik* również przyjmuje wartość **True**. Jeśli *wyrażenie1* ma wartość **True**, a *wyrażenie2* wartość **False**, wartością *wyniku* będzie **False**. Jeśli *wyrażenie1* przyjmuje wartość **False**, a *wyrażenie2* wartość **True**, wartością *wyniku* będzie **False**. W tabeli znajdującej się poniżej przedstawiono zależność wyniku od wartości wyrażeń.

Jeśli wyrażenie1 ma wartość	A wyrażenie2	Wartość wyniku wynosi
True	True	True
True	False	False
False	True	False
False	False	False

Gdy zastosujemy operator **And** wraz z wartościami numerycznymi, to wykona on porównanie na poziomie bitowym położonych w tych samych miejscach bitów w dwóch wyrażeniach numerycznych i ustawi w wyniku odpowiedni bit zgodnie z założeniami przedstawionymi w poniższej tabeli.

Jeśli bit w wyrażeniu1 ma wartość	A bit w wyrażeniu2 ma wartość	Wartość wyniku wynosi
0	0	0
0	1	0
1	0	0
1	1	1



Ponieważ operatory logiczne (bitowe) mają niższy priorytet niż operatory arytmetyczne czy operatory relacyjne, w celu zapewnienia poprawnego wykonywania działań wszystkie operacje bitowe powinny być ujęte w nawiasach.

Jeśli operandy składają się z jednego wyrażenia logicznego i jednego wyrażenia numerycznego, to wynik wyrażenia logicznego zostanie zamieniony na wartość numeryczną (–1 dla wartości **True**, a 0 dla wartości **False**) i zostanie przeprowadzona operacja na poziomie bitowym.

Przykład

W przykładzie wykorzystano operator **And** do wyliczenia iloczynu logicznego dwóch wyrażeń. Wynik jest wartością logiczną, która informuje o prawdziwości całego wyrażenia.

```
Dim A As Integer = 10
Dim B As Integer = 8
Dim C As Integer = 6
Dim mojTest As Boolean
mojTest = A > B And B > C ' Zwraca wartość True.
mojTest = B > A And B > C ' Zwraca wartość False.
```

Operator **And** został użyty do wyliczenia iloczynu logicznego pojedynczych bitów z dwóch wyrażeń numerycznych. Bit wyniku zostaje ustawiony, gdy odpowiadające mu bity w obu operandach mają wartość jeden.

```
Dim A As Integer = 10
Dim B As Integer = 8
Dim C As Integer = 6
Dim mojTest As Integer
mojTest = (A And B) ' Zwraca wartość 8.
mojTest = (A And C) ' Zwraca wartość 2.
mojTest = (B And C) ' Zwraca wartość 0.
```

Zobacz także:

- ◆ Operatory logiczne (bitowe);
- ◆ Priorytet operatorów w języku Visual Basic;
- ◆ Podział operatorów ze względu na funkcjonalność;
- ◆ Operator AndAlso;
- ◆ Operatory logiczne.

AndAlso — operator

Wylicza skrócony (ang. *short-circuit*) iloczyn logiczny dwóch wyrażeń.

wynik = wyrażenie1 AndAlso wyrażenie2

Części

wynik

Wymagana. Dowolne wyrażenie logiczne. Wynik jest wartością logiczną wynikającą z iloczynu logicznego dwóch wyrażeń.

wyrażenie1

Wymagana. Dowolne wyrażenie logiczne.

wyrażenie2

Wymagana. Dowolne wyrażenie logiczne.

Komentarze

Jeśli zarówno *wyrażenie1*, jak i *wyrażenie2* przyjmuje wartość **True**, to *wynik* ma wartość **True**. Jeśli *wyrażenie1* ma wartość **True**, a *wyrażenie2* ma wartość **False**, wartością *wyniku* będzie wartość **False**. Jeśli *wyrażenie1* przyjmuje wartość **False**, to wartość *wyrażenia2* nie będzie wyliczana, a *wynikowi* zostanie przypisana wartość **False** (operator skracia ocenę wartości wyrażenia). Poniższa tabela przedstawia sposób określania wartości *wyniku*.

Jeśli wyrażenie1 ma wartość	A wyrażenie2 ma wartość	Wartość wyniku wynosi
True	True	True
True	False	False
False	(nie jest wyliczane)	False

Przykład

W przykładzie wykorzystano operator **AndAlso** do wyliczenia iloczynu logicznego dwóch wyrażeń. Wynik jest wartością logiczną informującą o prawdziwości całego wyrażenia. Jeśli wartość pierwszego wyrażenia wynosi **False**, drugie wyrażenie w ogóle nie jest wyliczane.

```
Dim A As Integer = 10
Dim B As Integer = 8
Dim C As Integer = 6
Dim mojTest As Boolean
mojTest = A > B AndAlso B > C ' zwraca wartość True.
mojTest = B > A AndAlso B > C ' zwraca wartość False. Drugie wyrażenie
                             ' nie jest wyliczane.
mojTest = A > B AndAlso C > B ' zwraca wartość True. Drugie wyrażenie
                             ' jest wyliczane.

' Przykład demonstruje wykorzystanie operatora AndAlso do przeszukiwania
' wartości tablic. Jeśli wartość I jest mniejsza lub równa górnej granicy tablicy,
' to następuje próba znalezienia w tablicy poszukiwanej wartości.

Dim I As Integer = 0
While I <= UBound(arr) AndAlso arr(I) <> SearchValue
    I += 1
End While
```

Zobacz także:

- ◆ Operatory logiczne (bitowe);
- ◆ Priorytet operatorów w języku Visual Basic;
- ◆ Podział operatorów ze względu na funkcjonalność;
- ◆ Operatory logiczne.

Ansi

Słowo kluczowe **Ansi** sygnalizuje, że łańcuchy są konwertowane na wartości zgodne ze standardami ANSI (Amerykańskiego Instytutu Normalizacyjnego — *American National Standards Institute*) bez względu na nazwę deklarowanej metody.

Słowo kluczowe **Ansi** stosowane jest w kontekście:

Polecenie `Declare`.

Zobacz także:

- ♦ Słowa kluczowe języka Visual Basic.

AppActivate — funkcja

Aktywuje okno aplikacji.

```
Public Overloads Sub AppActivate( _
    ByVal { Tytuł As String | IDprocesu As Integer } _
)
```

Parametry

Tytuł

Wyrażenie typu **String**, określające tytuł pojawiający się na pasku tytułu okna aplikacji, które chcemy aktywować. Można w tym miejscu użyć identyfikatora zadania zwróconego przez funkcję **Shell**.

IDprocesu

Liczba całkowita określająca numer identyfikatora procesu (zdefiniowany przez Win32[®]), który jest przypisany do tego procesu.

Wyjątki i błędy

Typ wyjątku	Numer błędu	Warunek
ArgumentException	5	Nie znaleziono <i>IDprocesu</i> .

Komentarze

Funkcja **AppActivate** powoduje, że nazwana aplikacja lub okno staje się oknem aktywnym; nie wpływa jednak na to, czy okno jest zminimalizowane czy wyświetlone w trybie pełnoekranowym. Wykonywana aplikacja przestaje być oknem aktywnym, gdy użytkownik zamknie ją lub przełączy się do innej aplikacji. W celu uruchomienia aplikacji i ustawienia stylu okna należy użyć funkcji **Shell**.

Funkcji **AppActivate** można używać wyłącznie z procesami posiadającymi okna. Większość tekstowych aplikacji Win32 nie posiada okien, co oznacza, że aplikacje te nie pojawiają się na liście procesów, którą przeszukuje funkcja **AppActivate**. Gdy aplikacja uruchamiana jest za pomocą innej aplikacji konsolowej, system tworzy dla niej osobny proces, a standardowe wyjście tej aplikacji przekierowuje do procesu konsolowego. W rezultacie, gdy zażądamy identyfikatora bieżącego procesu, otrzymujemy identyfikator nowo utworzonego procesu, a nie identyfikator procesu okna konsoli.

W czasie uruchomienia funkcja **AppActivate** aktywuje dowolną działającą aplikację, której tytuł zgadza się z *Tytułem* lub tę, której identyfikator procesu jest zgodny z *ID-procesu*. Jeśli nie istnieje zgodność z żadną aplikacją, zostaje aktywowana aplikacja o tytule zaczynającym się podanym *Tytułem*. Jeśli istnieje więcej aplikacji nazwanych przy użyciu *Tytułu*, funkcja **AppActivate** aktywuje jedną — wybraną przypadkowo.

Przykład

Przykład przedstawia różne zastosowania funkcji **AppActivate** do aktywowania okna aplikacji. Procedury **Shell** zakładają, że aplikacje znajdują się w miejscach wskazanych przez ścieżki dostępu.

```
Dim notatnikID As Integer
' Aktywuje uruchomiony proces Notatnika.
AppActivate("Bez tytułu - Notepad")
' Funkcja AppActivate może także użyć wartości zwróconej przez funkcję Shell.
' Funkcja Shell uruchamia nowy egzemplarz notatnika.
notatnikID = Shell("C:\WINNT\NOTEPAD.EXE", AppWinStyle.MinimizedNoFocus)
' Aktywuje nowy egzemplarz notatnika.
AppActivate(notatnikID)
```

Zobacz także:

- ◆ Funkcja Shell.

AppWinStyle — wyliczenie

Używając poleceń **Shell**, zamiast rzeczywistych wartości można w kodzie użyć wymienionych poniżej składowych wyliczenia.

Składowe wyliczenia AppWinStyle

Argument *Styl* przyjmuje wymienione poniżej składowe wyliczenia **AppWinStyle**:

Składowa	Stała	Opis
Hide	vbHide	Okno jest ukryte i staje się aktywne.
NormalFocus	vbNormalFocus	Okno staje się aktywne i ma przywróconą oryginalną wielkość i pozycję.
MinimizedFocus	vbMinimizedFocus	Okno jest wyświetlone jako aktywna ikona na pasku zadań.

Składowa	Stała	Opis
MaximizedFocus	vbMaximizedFocus	Okno zostaje zmaksymalizowane i staje się aktywne.
NormalNoFocus	vbNormalNoFocus	Przywrócenie oknu ostatniego rozmiaru i pozycji. Okno, które było w danym momencie aktywne, pozostaje oknem aktywnym.
MinimizedNoFocus	vbMinimizedNoFocus	Okno jest wyświetlone jako ikona. Okno, które było w danym momencie aktywne, pozostaje oknem aktywnym.

Zobacz także:

- ♦ Funkcja Shell.

As

Słowo kluczowe **As** rozpoczyna klauzulę **As**, identyfikującą typ danych.

Słowo kluczowe **As** używane jest w kontekście:

Polecenie Const;	Polecenie Event;
Polecenie Declare;	Polecenie Function;
Polecenie Delegate;	Polecenie Property;
Polecenie Dim;	Polecenie Sub;
Polecenie Enum;	Polecenia Try...Catch...Finally.

Zobacz także:

- ♦ Słowa kluczowe języka Visual Basic.

Asc, AscW — funkcje

Zwracają wartości typu **Integer** będące kodem znaku przekazanego jako parametr.

```
Public Overloads Function Asc(ByVal łańcuch As Char) As Integer
Public Overloads Function AscW(ByVal łańcuch As Char) As Integer
```

-lub-

```
Public Overloads Function Asc(ByVal łańcuch As String) As Integer
Public Overloads Function AscW(ByVal łańcuch As String) As Integer
```

Parametr

Łańcuch

Wymagany. Dowolne poprawne wyrażenie łańcuchowe (**String**) lub znak (**Char**). Jeśli *Łańcuch* jest wyrażeniem łańcuchowym, to jako parametr zostanie wykorzystany wyłącznie pierwszy znak łańcucha. Jeśli *Łańcuch* nie będzie zawierał żadnych znaków lub przyjmie wartość **Nothing**, wystąpi błąd **ArgumentException**.

Wyjątki i błędy

Typ wyjątku	Numer błędu	Warunek
ArgumentException	5	Łańcuch nie został określony lub ma zerową długość.

Komentarze

Funkcja **Asc** zwraca *punkt kodowy* znaku wpisanego jako parametr, który jest także zwany kodem znaku. Może to być liczba z zakresu od 0 do 255 dla wartości jednobajтового zestawu znaków (*SBCS* — *single-byte character set*) lub liczba z zakresu –32768 do 32767 dla wartości dwubajтового zestawu znaków (*DBCS* — *double-byte character set*). Zwrócona wartość zależy od strony kodowej aktualnego wątku, zawartej we właściwości **ANSICodePage** klasy **TextInfo**. **TextInfo.ANSICodePage** można uzyskać poprzez **System.Globalization.CultureInfo.CurrentCulture.TextInfo.ANSICodePage**.

Funkcja **AscW** zwraca kod Unicode podanego znaku. Może to być wartość od 0 do 65535. Zwrócona wartość jest niezależna od ustawień kulturowych i strony kodowej danego wątku.

Przykład

W poniższym przykładzie funkcja **Asc** zwraca wartości typu **Integer** przedstawiające kod znaku odpowiadający pierwszej literze każdego z łańcuchów.

```
Dim MojaLiczba As Integer
MojaLiczba = Asc("A")      ' Przypisuje zmiennej MojaLiczba wartość 65.
MojaLiczba = Asc("a")      ' Przypisuje zmiennej MojaLiczba wartość 97.
MojaLiczba = Asc("Ala")    ' Przypisuje zmiennej MojaLiczba wartość 65.
```

Zobacz także:

- ◆ Funkcje **Chr**, **ChrW**;
- ◆ Funkcje konwersji;
- ◆ Funkcje konwersji typu.

Assembly

Słowo kluczowe **Assembly** sygnalizuje, że blok atrybutu na początku pliku źródłowego jest stosowany do całego komponentu, a nie tylko do aktualnego modułu.

Auto

Słowo kluczowe **Auto** sygnalizuje, że łańcuchy są konwertowane zgodnie z zasadami wspólnego środowiska uruchomieniowego na podstawie nazwy deklarowanej metody.

Słowo kluczowe **Auto** używane jest w kontekście:

Polecenie Declare.

Zobacz także:

- ♦ Słowa kluczowe języka Visual Basic.

Beep — funkcja

Powoduje wygenerowanie dźwięku przez głośnik komputera.

```
Public Sub Beep()
```

Komentarze

Wysokość dźwięku oraz czas trwania dźwięku zależą od sprzętu i oprogramowania systemowego, dlatego mogą być różne na różnych komputerach.

Przykład

W tym przykładzie użyto funkcji **Beep** do wysłania długiego, nieprzerwanego dźwięku poprzez głośniki komputera.

```
Dim I As Integer
For I = 1 To 100 ' Pętla wykonywana 100 razy.
    Beep          ' Wydaje dźwięk.
Next I
```

Zobacz także:

- ♦ Części składowe biblioteki wykonawczej Visual Basic.

Boolean — typ danych

Zmienne logiczne przechowywane są jako liczby 16-bitowe (2-bajtowe), ale mogą mieć wyłącznie wartości prawda (**True**) lub fałsz (**False**). W celu przypisania do zmiennych logicznych jednego z tych dwóch stanów należy używać słów kluczowych **True** i **False**.

Gdy dane numeryczne są konwertowane na wartości logiczne, 0 staje się wartością **False**, a wszystkie inne wartości zostają zamienione na **True**. Gdy wartości logiczne konwertowane są na wartości numeryczne, z wartości **False** powstaje 0, z wartości **True** powstaje -1.



Nie powinno się pisać kodu zależnego od wartości numerycznych, które odpowiadają wartościom **True** i **False**. Gdy tylko jest to możliwe, powinno się ograniczyć użycie zmiennych typu **Boolean** tylko do wartości logicznych, dla których zostały one przeznaczone. Jeśli konieczne jest mieszanie wartości logicznych i numerycznych, należy użyć odpowiednich słów kluczowych konwersji.

Odpowiednikiem tego typu danych na platformie .NET jest **System.Boolean**.

Zobacz także:

- ♦ Typy danych — podsumowanie;
- ♦ Typ danych Integer;
- ♦ Funkcje konwersji typu;
- ♦ Konwersja — podsumowanie.

ByRef

Słowo kluczowe **ByRef** sygnalizuje sposób przekazywania argumentów, w efekcie którego wywoływana procedura może zmienić wartość zmiennej, będącej argumentem w kodzie wywołującym.

Słowo kluczowe **ByRef** używane jest w kontekście:

Polecenie Declare;
Polecenie Function;
Polecenie Sub.

Zobacz także:

- ♦ Słowa kluczowe języka Visual Basic.

Byte — typ danych

Zmienne typu **Byte** przechowywane są jako 8-bitowe (1-bajtowe) liczby bez znaku z zakresu od 0 do 255.

Typ danych **Byte** służy do przechowywania danych binarnych.



Typ danych **Byte** można przekształcać bez wystąpienia błędu **System.OverflowException** na jeden z następujących typów danych: **Short**, **Integer**, **Long**, **Single**, **Double** lub **Decimal**.

Odpowiednikiem tego typu danych na platformie .NET jest **System.Byte**.

Zobacz także:

- ◆ Typy danych — podsumowanie;
- ◆ Typ danych Integer;
- ◆ Typ danych Short;
- ◆ Funkcje konwersji typu;
- ◆ Konwersja — podsumowanie.

ByVal

Słowo kluczowe **ByVal** sygnalizuje sposób przekazywania argumentów, w efekcie którego wywoływana procedura nie może zmienić wartości zmiennej będącej argumentem w kodzie wywołującym.

Słowo kluczowe **ByVal** używane jest w kontekście:

- Polecenie Declare;
- Polecenie Function;
- Polecenie Sub.

Zobacz także:

- ◆ Słowa kluczowe języka Visual Basic.

Call — polecenie

Przekazuje sterowanie do procedury **Sub**, do procedury **Function** lub do procedury biblioteki dołączanej dynamicznie (*DLL* — *dynamic-link library*).

[Call] NazwaProcedury[(ListaArgumentów)]
--

Części

NazwaProcedury

Wymagana. Nazwa procedury, która ma zostać wywołana.

ListaArgumentów

Opcjonalna. Lista zmiennych lub wyrażeń reprezentujących argumenty przekazywane do procedury, gdy ta jest wywoływana. Argumenty oddzielane są przecinkami. Jeśli dołączamy *ListęArgumentów*, należy umieścić ją w nawiasach.

Komentarze

Przy wywoływaniu procedury nie jest wymagane użycie słowa kluczowego **Call**. Jeśli zostanie ono użyte do wywołania wewnętrznej funkcji języka Visual Basic, biblioteki DLL czy funkcji zdefiniowanej przez użytkownika, wartość zwracana przez funkcję zostanie odrzucona.

Przykład

Przykład ilustruje użycie polecenia **Call** do przekazania kontroli do procedury **Sub**, do funkcji wewnętrznej i do procedury biblioteki dołączanej dynamicznie (DLL).

```
' Wywołanie procedury Sub.
Call PrintToDebugWindow("Witaj świecie")
' Powyższe polecenie przekazuje kontrolę do poniższej procedury Sub.
Sub PrintToDebugWindow(ByVal AnyString As String)
    Debug.WriteLine(AnyString) ' Wyświetla łańcuch w oknie Output.
End Sub

' Wywołanie funkcji wewnętrznej. Zwracana wartość jest odrzucona.
Call Shell("C:\Windows\calc.exe", AppWinStyle.NormalFocus)

' Wywołuje procedurę DLL Microsoft Windows. Polecenie Declare musi być
' prywatne w klasie, nie w module.
Private Declare Sub MessageBeep Lib "User" (ByVal N As Integer)
Sub CallMyDll()
    Call MessageBeep(0) ' Wywołuje procedurę DLL Windows.
    MessageBeep(0) ' Wywołuje ponownie, bez słowa kluczowego Call.
End Sub
```

Zobacz także:

- ◆ Polecenie Declare;
- ◆ Polecenie Function;
- ◆ Polecenie Sub.

CallByName — funkcja

Wykonuje metodę obiektu lub ustawia bądź zwraca właściwość obiektu.

```
Public Function CallByName( _
    ByVal Obiekt As System.Object, _
    ByVal NazwaProc As String, _
    ByVal TypWywołania As CallType, _
    ByVal ArgumParamArray() As Object _
) As Object
```

Parametry

Obiekt

Wymagany. Obiekt. Wskaźnik obiektu udostępniającego właściwość lub metodę.

NazwaProc

Wymagany. Łańcuch. Wyrażenie łańcuchowe zawierające nazwę właściwości lub metody obiektu.

TypWywołania

Wymagany. Składowa wyliczenia typu **Microsoft.VisualBasic.CallType** wskazująca na typ wywoływanej procedury. Wartościami typu wywołania (**CallType**) mogą być **Method**, **Get** lub **Set**.

ArgumParamArray()

Opcjonalny. **ParamArray**. Tablica parametrów zawierająca argumenty, które mają być przekazane do wywoływanej metody lub właściwości.

Wyjątki i błędy

Typ wyjątku	Numer błędu	Warunek
ArgumentException	5	Niepoprawna wartość <i>TypWywołania</i> ; musi przyjąć wartość Method , Get lub Set .

Komentarze

Funkcja **CallByName** wykorzystywana jest podczas działania programu do pobrania lub ustawienia właściwości lub do wywołania metody przy użyciu łańcucha określającego nazwę właściwości lub metody.

Przykład

W pierwszej linii przykładu funkcję **CallByName** wykorzystano do ustawienia właściwości **Text** pola tekstowego. Druga linia pobiera wartość właściwości **Text**, trzecia linia wywołuje metodę **Move**, aby przesunąć pole tekstowe.

```
Imports Microsoft.VisualBasic.CallType
' Importowane instrukcje muszą znajdować się na początku modułu.
...
Sub TestujCallByName1()
' Ustawienie właściwości.
  CallByName(TextBox1, "Text", CallType.Set, "Nowy Tekst")
' Wczytuje wartość właściwości.
  MsgBox(CallByName(TextBox1, "Text", CallType.Get))
' Wywołuje metodę.
  CallByName(TextBox1, "Hide", CallType.Method)
End Sub
```

Kolejny przykład wykorzystuje funkcję **CallByName** do wywołania metod **Add** i **Item** obiektu kolekcji.

```
Public Sub TestujCallByName2()
  Dim kolekcja As New Collection()
  ' Zapisuje łańcuch "Pierwszy element" w kolekcji przez
  ' wywołanie metody Add.
```

```

    CallByName(kolekcja, "Add", CallType.Method, "Pierwszy element")
' Wczytuje pierwszą pozycję kolekcji używając
' właściwości Item i wyświetla ją za pomocą MsgBox().
    MsgBox(CallByName(kolekcja, "Item", CallType.Get, 1))
End Sub

```

Zobacz także:

- ♦ Wyliczenie CallType;
- ♦ Tablice parametrów.

CallType — wyliczenie

Wywołując funkcję **CallByName** można zamiast faktycznych wartości użyć odpowiednich składowych wyliczenia **CallType**.

Składowe wyliczenia CallType

Argument *TypWywołania* przyjmuje wymienione poniżej wartości **CallType**:

Składowa	Stała	Opis
Method	vbMethod	Wywołanie metody.
Get	vbGet	Wczytanie wartości właściwości.
Set	vbSet	Ustawienie wartości właściwości.

Zobacz także:

- ♦ Funkcja CallByName.

Case

Słowo kluczowe **Case** wprowadza wartość lub zestaw wartości, z którymi ma być porównana wartość wyrażenia.

Słowo kluczowe **Case** używane jest w kontekście:

Polecenia Select...Case.

Zobacz także:

- ♦ Słowa kluczowe języka Visual Basic.

Char — typ danych

Zmienne typu **Char** przechowywane są jako 16-bitowe (2-bajtowe) liczby bez znaku, z zakresu od 0 do 65535. Każda liczba reprezentuje jeden znak Unicode. Bezpośrednia konwersja między typem danych **Char** a typem numerycznym nie jest możliwa, ale można w tym celu posłużyć się funkcjami **AscW** i **ChrW**.

Dołączenie znacznika typu **C** do jednoliterowego łańcucha wymusza użycie typu danych **Char**. Użycie znacznika typu jest wymagane, gdy włączone jest sprawdzanie typu (polecenie **Option Strict**), tak jak ma to miejsce w następującym przykładzie:

```
Option Strict On
' ...
Dim ZmieZnak As Char
ZmieZnak = "Z" ' Nie może przekształcić łańcucha na typ Char,
               ' gdy włączone jest sprawdzanie typu.
ZmieZnak = "Z"C ' Pomyślnie przypisuje pojedynczy znak zmiennej ZmieZnak.
```

Odpowiednikiem tego typu danych na platformie .NET jest **System.Char**.

Zobacz także:

- ◆ Typy danych — podsumowanie;
- ◆ Typ danych Integer;
- ◆ Funkcje konwersji typu;
- ◆ Konwersja — podsumowanie;
- ◆ Funkcje Asc, AscW;
- ◆ Funkcje Chr, ChrW.

ChDir — funkcja

Zmienia bieżący katalog lub folder.

```
Public Sub ChDir(ByVal Ścieżka As String)
```

Parametr

Ścieżka

Wymagany. Wyrażenie łańcuchowe identyfikujące katalog lub folder, który stanie się nowym katalogiem lub folderem domyślnym. *Ścieżka* może zawierać literowe oznaczenie dysku. Jeśli dysk nie jest podany, funkcja **ChDir** zmienia domyślny katalog lub folder na dysku bieżącym.

Wyjątki i błędy

Typ wyjątku	Numer błędu	Warunek
<code>ArgumentException</code>	52	Parametr <i>Ścieżka</i> jest pusty.
<code>FileNotFoundException</code>	76	Podany jest nieodpowiedni dysk lub dysk jest niedostępny.

Komentarze

Funkcja **ChDir** zmienia domyślny katalog, ale nie zmienia domyślnego dysku. Jeśli na przykład domyślnym dyskiem jest C, to poniższe polecenie zmieni domyślny katalog na dysku D, ale C nadal pozostanie dyskiem domyślnym:

```
ChDir "D:\TMP"
```

Zmiany katalogu można dokonać również względem bieżącego katalogu używając "..", tak jak ma to miejsce poniżej:

```
ChDir ".." ' Przenosi o jeden katalog w górę.
```

Przykład

W tym przykładzie funkcję **ChDir** wykorzystano do zmiany bieżącego katalogu lub folderu.

```
' Zmienia bieżący katalog lub folder na "MojKat".
ChDir("MojKat")
```

```
' Załóżmy, że "C:" jest dyskiem bieżącym. Następująca instrukcja zmieni
' domyślny katalog na dysku "D:". "C:" pozostanie dyskiem bieżącym.
ChDir("D:\WINDOWS\SYSTEM")
```

Zobacz także:

- ◆ Funkcja ChDrive;
- ◆ Funkcja CurDir;
- ◆ Funkcja Dir;
- ◆ Funkcja Mkdir;
- ◆ Funkcja Rmdir.

ChDrive — funkcja

Zmienia bieżący dysk.

```
Public Overloads Sub ChDrive(ByVal Dysk As { Char | String })
```

Parametr*Dysk*

Wymagany. Wyrażenie łańcuchowe podające istniejący dysk. Jeśli zostanie podany łańcuch zerowej długości (""), aktualny dysk nie zostanie zmieniony. Jeśli argument *Dysk* jest łańcuchem kilkunastokrotnym, funkcja **ChDrive** użyje tylko pierwszej litery.

Wyjątki i błędy

Typ wyjątku	Numer błędu	Warunek
IOException	68	Podano nieodpowiedni dysk lub dysk jest niedostępny.

Przykład

Pokazano użycie funkcji **ChDrive** do zmiany bieżącego dysku.

```
ChDrive("D") ' Powoduje, że bieżącym dyskiem staje się dysk "D".
```

Zobacz także:

- ♦ Funkcja ChDir;
- ♦ Funkcja CurDir;
- ♦ Funkcja Mkdir;
- ♦ Funkcja Rmdir.

Choose — funkcja

Wybiera i zwraca wartość z listy argumentów.

```
Public Function Choose( _
    ByVal Indeks As Double, _
    ByVal ParamArray Wybór() As Object _
) As Object
```

Parametry*Indeks*

Wymagany. Typ **Double**. Wyrażenie numeryczne, które jest wartością z przedziału od 1 do wartości równej liczbie elementów przekazanych w argumentcie *Wybór*.

Wybór()

Wymagany. Tablica parametrów typu **Object**. Można podać pojedynczą zmienną lub wyrażenie typu **Object**, listę wyrażeń lub zmiennych typu **Object**, oddzielonych przecinkami lub jednowymiarową tablicę elementów typu **Object**.

Wyjątki i błędy

Typ wyjątku	Numer błędu	Warunek
ArgumentException	5	Ilość wymiarów tablicy <i>Wybór()</i> $\leq$ 1.

Komentarze

Funkcja **Choose** zwraca element listy przekazanej przez *Wybór()*, bazując na wartości *Indeksu*. Gdy *Indeks* ma wartość 1, wybierany jest pierwszy element listy. Ostatni element listy zostanie wybrany, gdy wartość *Indeks* będzie równa **UBound**(*Wybór()*). Jeśli *Indeks* wykracza poza ten limit, funkcja **Choose** zwraca **Nothing**.

Jeśli *Indeks* nie jest liczbą całkowitą, zanim zostanie oszacowany, będzie zaokrąglony do najbliższej liczby całkowitej.

Funkcji **Choose** można użyć do wyszukania wartości z listy możliwości.



Wyrażenia znajdujące się na liście argumentów mogą zawierać wywołania funkcji. Zanim zostanie wywołana funkcja **Choose**, kompilator wywołuje każdą funkcję w każdym wyrażeniu w ramach przygotowania listy argumentów do wywołania funkcji. Nie należy więc oczekiwać, że nie zostanie wywołana funkcja niewybrana przez wartość argumentu *Indeks*.

Przykład

Przykład pokazuje wykorzystanie funkcji **Choose** do wyświetlenia nazwy w odpowiedzi na przekazaną procedurze wartość indeksu w parametrze *Indeks*.

```
Function Wybor(Indeks As Integer) As String
    GetChoice = CStr(Choose(Indeks, "DHL", "Servisco", "Pocztex"))
End Function
```

Zobacz także:

- ♦ Funkcja IIf;
- ♦ Polecenia Select...Case;
- ♦ Funkcja Switch.

Chr, ChrW — funkcje

Zwraca znak skojarzony z podanym kodem znaku.

```
Public Function Chr(ByVal KodZnaku As Integer) As Char
Public Function ChrW(ByVal KodZnaku As Integer) As Char
```

Parametr

KodZnaku

Wymagany. Wyrażenie typu **Integer** odpowiadające *punktowi kodowemu* danego znaku (zwanemu inaczej kodem znaku). Jeśli *KodZnaku* wykracza poza zakres z przedziału od -32768 do 65535 , występuje błąd **ArgumentException**.

Wyjątki i błędy

Typ wyjątku	Numer błędu	Warunek
ArgumentException	5	<i>KodZnaku</i> < -32768 lub > 65535 .

Komentarze

Niesymetryczny zakres dopuszczalnych wartości parametru *KodZnaku* kompensuje różnice w zapisie, występujące pomiędzy typami danych **Short** i **Integer**. Na przykład &H8E01 jako typ danych **Short** to -29183 , a jako typ danych **Integer** to $+36353$. Zapewnia to też zgodność z Visual Basic 6.0.

Funkcja **Chr** korzysta z klasy **Encoding** w przestrzeni nazw **System.Text**, aby określić, czy aktualny wątek używa jednobajtowego zestawu znaków (SBCS) czy dwubajtowego zestawu znaków (DBCS). Następnie interpretuje *KodZnaku* jako punkt kodowy w odpowiednim zestawie. Zakres może wynosić od 0 do 255 dla znaków z jednobajtowego zestawu znaków lub od -32768 do 65535 dla znaków z dwubajtowego zestawu znaków. Zwrócony znak zależy od strony kodowej aktualnego wątku, która jest określona przez właściwość **ANSICodePage** klasy **TextInfo**. Wartość **TextInfo.ANSICodePage** można uzyskać poprzez podanie **System.Globalization.CultureInfo.CurrentCulture.TextInfo.ANSICodePage**.

Funkcja **ChrW** przyjmuje *KodZnaku* jako punkt kodowy Unicode. Zakres jest niezależny od ustawień regionalnych oraz strony kodowej aktualnego wątku. Wartości z zakresu od -32768 do -1 traktowane są tak samo jak wartości z zakresu $+32768$ do $+65535$.

Liczby (kody znaków) od 0 do 31 są tym samym co standardowe, niedrukowalne znaki kodu ASCII. Na przykład **Chr**(10) zwraca znak przejścia do nowej linii.

Przykład

Przedstawia użycie funkcji **Chr** w celu uzyskania znaku skojarzonego z podanym kodem znaku.

```
Dim MojZnak As Char
MojZnak = Chr(65) ' Zwraca "A".
MojZnak = Chr(97) ' Zwraca "a".
MojZnak = Chr(62) ' Zwraca ">".
MojZnak = Chr(37) ' Zwraca "%".
```

Zobacz także:

- ◆ Funkcje **Asc**, **AscW**;
- ◆ Funkcja **Str**;

- ◆ Funkcje konwersji;
- ◆ Funkcje konwersji typu.

Class — polecenie

Deklaruje nazwę klasy, a także definicję zmiennych, właściwości, zdarzeń i metod tej klasy.

```
[ <ListaAtrybutów> ] [ Public | Private | Protected | Friend |  
Protected Friend ] [ Shadows ] [ MustInherit | NotInheritable ] _  
Class nazwa  
    [ Inherits nazwaKlasy ]  
    [ Implements nazwyInterfejsów ]  
    [ instrukcje ]  
End Class
```

Części

ListaAtrybutów

Opcjonalna. Lista atrybutów stosowanych do klasy. Kolejne atrybuty oddziela się przecinkami.

Public

Opcjonalna. Elementy zadeklarowane przy użyciu modyfikatora **Public** posiadają dostęp publiczny. Nie ma restrykcji dotyczących użycia elementów publicznych.

Private

Opcjonalna. Elementy zadeklarowane przy użyciu modyfikatora **Private** posiadają dostęp prywatny. Elementy prywatne są dostępne wyłącznie z wnętrza kontekstu deklaracji (włączając w to obiekty w nim zagnieżdżone).

Protected

Opcjonalna. Elementy zadeklarowane przy użyciu modyfikatora **Protected** posiadają dostęp chroniony. Są dostępne wyłącznie z wnętrza ich własnej klasy lub z klasy pochodnej. Dostęp chroniony może być określony tylko dla składowych klas. Dostęp zaprzyjaźniony nie jest szczególnym przypadkiem dostępu chronionego.

Friend

Opcjonalna. Elementy zadeklarowane przy użyciu modyfikatora **Friend** posiadają dostęp zaprzyjaźniony. Element o dostępie zaprzyjaźnionym jest dostępny wyłącznie z wnętrza programu zawierającego deklarację tego obiektu. Klasy, które nie mają podanego modyfikatora dostępu, deklarowane są domyślnie jako zaprzyjaźnione.

Protected Friend

Opcjonalna. Elementy zadeklarowane przy użyciu modyfikatorów **Protected Friend** posiadają dostęp, będący połączeniem dostępu chronionego i zaprzyjaźnionego.

Shadows

Opcjonalna. Sygnalizuje, że klasa zakrywa identycznie nazwany element w klasie bazowej. Można zakryć dowolny zadeklarowany element elementem innego typu. Zakrywany element nie jest dostępny w klasach pochodnych, które go zakrywają.

MustInherit

Opcjonalna. Sygnalizuje, że niewspółużytkowane elementy składowe klasy są dostępne tylko poprzez klasy pochodne. Nie można tworzyć egzemplarzy klas z wymuszonym dziedziczeniem.

NotInheritable

Opcjonalna. Sygnalizuje, że od danej klasy nie jest dozwolone żadne dziedziczenie.

nazwa

Wymagana. Nazwa klasy. Stosują się do niej standardowe zasady nazywania zmiennych.

Inherits

Opcjonalna. Sygnalizuje, że dana klasa dziedziczy składowe innej klasy. Klasa może dziedziczyć tylko od jednej klasy.

nazwaKlasy

Nazwa klasy bazowej, od której dziedziczy dana klasa.

Implements

Opcjonalna. Sygnalizuje, że dana klasa implementuje składowe interfejsu. Jeśli jest użyte polecenie **Implements**, to musi ono występować bezpośrednio za poleceniami **Inherits** po poleceniu **Class** i musi implementować każdą składową zdefiniowaną przez każdy podany interfejs.

nazwyInterfejsów

Wymagana, jeśli zostało użyte polecenie **Implements**. Nazwa interfejsu implementowanego przez daną klasę.

Instrukcje

Opcjonalna. Polecenia, definiujące zmienne, właściwości, zdarzenia, metody i zagnieżdżone typy klasy.

End Class

Kończy blok klasy.

Każdy atrybut w części *ListaAtrybutów* ma przedstawione niżej części i składnię.

<i>nazwaAtrybutu</i> [({ <i>argAtrybutu</i> <i>inicAtrybutu</i> })]

Części listyAtrybutów

nazwaAtrybutu

Wymagana. Nazwa atrybutu. Musi być poprawnym identyfikatorem języka Visual Basic.

argAtrybutu

Opcjonalna. Lista argumentów pozycyjnych danego atrybutu. Jeśli występuje większa ilość argumentów, oddziela się je przecinkami.

inicAtrybutu

Opcjonalna. Lista inicjatorów pól lub właściwości danego atrybutu. Większa ilość argumentów oddzielana jest przecinkami.

Komentarze

Klasy, przy których nie został podany modyfikator dostępu, mają domyślnie deklarowany dostęp typu **Friend**. Wewnątrz bloku **Class** składowe deklarowane są przy użyciu odpowiednich poleceń deklarujących jako **Public**, **Private**, **Protected**, **Friend** lub **Protected Friend**. Elementy zadeklarowane jako prywatne (**Private**) są widoczne wyłącznie wewnątrz bloku klasy. Elementy zadeklarowane jako **Public** są widoczne wewnątrz bloku klasy, a także są widoczne dla kodu spoza bloku klasy. Elementy, którym nie przypisano modyfikatora dostępu, domyślnie deklarowane są jako **Public**; wyjątkiem są stałe i pola, które domyślnie deklarowane są jako **Private**. Zmienne publiczne, zwane również polami, służą jako właściwości klasy — tak samo jak właściwości jawnie zadeklarowane za pomocą deklaracji **Property**. Domyślne właściwości i metody danej klasy są określane w deklaracji za pomocą słowa kluczowego **Default**. Więcej informacji na temat zastosowania tego słowa kluczowego można znaleźć w tematach poświęconych poszczególnym poleceniom deklaracji.

Dołączenie nieuściślonych nazw w zagnieżdżonych klasach spowoduje przeszukanie składowych danej klasy, następnie składowych klasy zawierającej ją i tak dalej, aż do najbardziej zewnętrznej klasy zawierającej. Można się odwoływać do prywatnych składowych klas zewnętrznych, ale przy odwołaniach do egzemplarzy składowych klasy zawierającej pojawi się błąd.

Zagnieżdżone klasy nie mogą dziedziczyć od klasy zawierającej je.

Przykład

W przykładzie wykorzystano polecenie **Class** do zdefiniowania klasy, w której mogą być stworzone zmienne, właściwości, metody i zdarzenia.

```
Public Class TaKlasa
    ' [deklaracje zmiennych, właściwości, metod i zdarzeń]
End Class
```

Zobacz także:

- ◆ Polecenie Inherits;
- ◆ Polecenie Implements;
- ◆ Polecenie Interface;
- ◆ Polecenie Property.