5

Les fonctions de Visual Basic.NET

5. Les fonctions de Visual Basic.NET

Visual Basic comprend un nombre important de fonctions permettant au programmeur de procéder à toutes sortes de traitements. Les fonctions retournent généralement des valeurs de retour et leur appel doit être suivi de parenthèses. Dans l'exemple suivant, la fonction `Now()` est utilisée afin de connaître la date et l'heure courantes. La valeur de retour sera stockée au sein de la variable `Aujourdhui`.

```
Dim Aujourdhui As Date
Aujourdhui = Today()
```

Certaines fonctions attendent des paramètres. Lorsque les fonctions attendent des paramètres, ceux-ci sont précisés entre les parenthèses et séparés les uns des autres par une virgule. Voici un exemple au sein de lequel la variable `Position` contiendra la position à laquelle l'expression *'planète'* est trouvée au sein d'une chaîne de caractères. Finalement, la fonction `MsgBox()` est utilisée afin d'afficher à l'utilisateur le résultat de la trouvaille :

```
Dim Position As Integer
Dim Chaine As String = "Allô la planète"
Position = InStr(Chaine, "planète")

MsgBox( "Planète trouvée à la position " & Position )
```

Les fonctions seront ici présentées au sein de tableau où la première colonne contient le nom de la fonction, la seconde colonne contient une description de la fonction et la troisième colonne contient un exemple de l'utilisation de la fonction. Notez que vous trouverez des explications très détaillées de chacune des fonctions dans le fichier d'aide MSDN de Visual Basic.NET.

Pour ceux et celles qui auraient travaillé à l'aide des versions antérieures de Visual Basic, notez qu'il est désormais obligatoire d'encadrer les paramètres au sein de parenthèse et, ce, même lorsque la valeur de retour de la fonction n'est pas retenue. Cette nouvelle syntaxe a été adoptée afin de standardiser l'appel des fonctions et procédures. Notez que l'éditeur de Visual Basic.NET inscrit automatiquement les parenthèses si vous les omettez.

Fonctions de conversion de types

Visual Basic prévoit les fonctions nécessaires afin de convertir explicitement des valeurs d'un type de données vers un type de données différent. Ces fonctions peuvent s'avérer grandement utiles lorsque vient le temps, par exemple, de traiter une date saisie par l'utilisateur au sein d'une boîte de texte et de calculer, par exemple, le nombre de jour la séparant de la date d'aujourd'hui. Dans ce cas, la chaîne de caractères contenues dans la boîte de texte doit être convertie explicitement en format `Date` avant d'être traitée. Voyez l'exemple suivant mais notez que les fonctions de traitement des dates seront vues en profondeur plus loin dans ce chapitre.

```
Dim MaDate As Date, NbrJours As Integer
MaDate = CDate(txtDate.Text)

NbrJours = DateDiff(DateInterval.Day, MaDate, Date())
```

Les fonctions de conversion de types trouvent toute leur utilité lorsque vous codez avec l'option `Strict` à la valeur `On`. Dans ce cas, tous les types de donnée doivent correspondre afin de pouvoir s'assigner mutuellement. S'ils ne correspondent pas, les fonctions de conversions de type doivent être utilisées afin de résoudre le problème.

Voici la liste des fonctions de conversion de type prévues par Visual Basic.NET :

Fonction	Signification	Exemple	
CBool	Conversion explicite en Boolean.	`CBool(2)`	Retourne True
CByte	Conversion explicite en Byte.	`CByte(1.2)`	Retourne 1
CChar	Conversion explicite en Char.	`CChar(65)`	
CDate	Conversion explicite en Date.	`CDate("2000-01-01")`	
CDbl	Conversion explicite en Double.	`CDbl(1.2)`	
CDec	Conversion explicite en Decimal.	`CDec(1.2)`	
CInt	Conversion explicite en Integer.	`CInt(1.2)`	Retourne 1
CLng	Conversion explicite en Long.	`CLng(1.2)`	Retourne 1
CObj	Conversion explicite en Objet.		
CShort	Conversion explicite en Short.	`CShort(1.2)`	Retourne 1
CSng	Conversion explicite en Single.		
CStr	Conversion explicite en String.	`CStr(stNom)`	
CType	Conversion explicite en type de donnée spécifié.	`CType(nbrA, Single)`	

Hex	Retourne la valeur hexadécimale de l'expression.	`Hex(32)`	Retourne 20
Oct	Retourne la valeur octale de l'expression.	`Oct(32)`	Retourne 40
Val	Retourne la valeur numérique de l'expression	`Val(nbrA)`	

Fonctions de types de données

Visual Basic prévoit certaines fonctions permettant de connaître le type d'information contenu au sein d'une variable avant de procéder à son traitement. Ces fonctions peuvent s'avérer utiles lorsque vient le temps, par exemple, de traiter une date saisie par l'utilisateur au sein d'une boîte de texte et de vérifier que l'utilisateur a saisi une valeur valide.

```
Dim dtNaissance As Date
If IsDate(txtNaissance.Text) Then
      dtNaissance = CInt(txtNaissance.Text)
Else
      MsgBox ("Veuillez entre une date valide !")
End If
```

Fonction	Signification	Exemple

IsArray	Retourne `True` si l'expression spécifiée est un tableau.	`Dim Tb() As Integer`
		`If IsArray(Tb) Then`
		`End If`

IsDate	Retourne `True` si l'expression spécifiée est une date valide.	`If IsDate(x) Then` `End If`
IsDBNull	Retourne `True` si l'expression spécifiée est une valeur nulle non-spécifiée correspondant au type `System.DBNull`. Utilisé principalement lors d'accès à des champs nuls au sein de bases de données.	`If IsDBNull(x) Then` `End If`
IsError	Retourne `True` si l'expression spécifiée est un objet de type `Error`.	`If IsError(obj) Then` `End If`
IsNothing	Retourne `True` si l'objet spécifié n'a pas été instancié ou a été détruit.	`If IsNothing(obj) Then` `End If`
IsNumeric	Retourne `True` si l'expression spécifiée est une valeur numérique valide.	`If IsNumeric(x) Then` `End If`

IsReference	Retourne `True` si l'expression spécifiée est une référence. Retourne donc `True` lorsque appliquée à des tableaux, des objets ou des chaînes de caractères.	`Dim Tb() As Integer`
		`If IsReference(Tb) Then`
		`End If`

SystemTypeName	Retourne une chaîne de caractères contenant le nom du type de données .NET correspondant au type de données Visual Basic spécifié.	`Dim Ss As String`
		`Ss = SystemTypeName(▼`
		`"Long")`
		`'Ss vaut "System.Int64"`

TypeName	Retourne une chaîne de caractères contenant le nom du type de données de la variable spécifiée.	`Dim Ss As String`
		`Dim X As Integer`
		`Ss = TypeName(x)`
		`'Ss vaut "Integer"`

VarType	Retourne le type de données de la variable spécifiée.	`Dim N As Integer` `X = VarType(N)`

vbTypeName	Retourne une chaîne de caractères contenant le nom du type de données Visual Basic correspondant au type de données .NET spécifié.	`Dim Ss As String`
		`Ss = SystemTypeName(▼`
		`"Int32")`
		`'Ss vaut "Integer"`

Fonctions d'interactivité avec l'utilisateur

Visual Basic prévoit quelques fonctions permettant l'interaction directe avec l'utilisateur dont deux que nous étudierons plus en profondeur.

Fonction	Description	Exemple
Beep	Génère un signal sonore.	`Beep()`
MsgBox	Affiche une boîte de message à l'utilisateur.	`MsgBox ("Allô la planète")`
InputBox	Affiche une boîte d'invite à l'utilisateur et retourne la valeur saisie par celui-ci.	`strR = InputBox("Entrez " ▼ & "un nombre entre 1 et 10")`

La fonction MsgBox

La fonction **MsgBox** permet de spécifier le texte à afficher au sein de la boîte de dialogue mais également les boutons et icônes à afficher ainsi que le titre que devra arborer la boîte de dialogue selon le prototype suivant :

```
MsgBox ( Message [, Boutons] [,Titre] )
```

Le paramètre *Boutons* peut prendre l'une des valeurs constantes suivantes :

Constante	Valeur	Description
MsgBoxStyle.OkOnly	0	Affiche le bouton Ok seulement.
MsgBoxStyle.OkCancel	1	Affiche les boutons Ok et Annuler.
MsgBoxStyle.AbortRetryIgnore	2	Affiche les boutons Abandonner, Réessayer et Ignorer.
MsgBoxStyle.YesNoCancel	3	Affiche les boutons Oui, Non et Annuler.
MsgBoxStyle.YesNo	4	Affiche les boutons Oui et Annuler.
MsgBoxStyle.RetryCancel	5	Affiche les boutons Réessayer et Annuler.
MsgBoxStyle.Critical	16	Affiche l'icône correspondant aux erreurs critiques.
MsgBoxStyle.Question	32	Affiche l'icône correspondant aux questions.
MsgBoxStyle.Exclamation	48	Affiche l'icône correspondant aux avertissements.
MsgBoxStyle.Information	64	Affiche l'icône correspondant aux informations.
MsgBoxStyle.DefaultButton1	0	Sélectionne par défaut le premier bouton.
MsgBoxStyle.DefaultButton2	256	Sélectionne par défaut le second bouton.
MsgBoxStyle.DefaultButton3	512	Sélectionne par défaut le troisième bouton.
MsgBoxStyle.SystemModal	4096	Suspend le système jusqu'à ce que l'utilisateur ait répondu à la boîte de message.
MsgBoxStyle.MsgBoxRight	524288	Affiche le texte aligné à droite.

Pour ceux et celles qui auraient travaillé à l'aide des versions antérieures de Visual Basic, notez que les anciennes constantes *(vbInformation, vbExclamation, etc)* sont encore disponibles et possèdent les mêmes valeurs que les nouvelles constantes .NET. Ces dernières seront utilisées au sein de cet ouvrage afin d'augmenter la compatibilité inter-langages du code.

Plusieurs valeurs constantes peuvent être combinées à l'aide de l'opérateur + comme le montre l'exemple suivant :

```
MsgBox ("Désirez-vous interrompre le processus ?", ▼
      MsgBoxStyle.YesNo + MsgBoxStyle.Question + ▼
      MsgBoxStyle.DefaultButton2 )
```

Ce qui produit le résultat ci-contre. Notez le titre de la boîte de dialogue, les deux boutons *Oui* et *Non* mais notez plus particulièrement que le focus a été déposé par défaut sur le second bouton.
On tentera toujours de déposer le focus par défaut sur le bouton dont la réponse est la moins dangereuse pour l'utilisateur.

Maintenant, reste à récupérer la réponse de l'utilisateur à une telle question. La fonction `MsgBox` permet de connaître la réponse de l'utilisateur en testant la valeur retournée par cette fonction :

Constante	Valeur	Description
MsgBoxResult.Ok	1	L'utilisateur a appuyé sur le bouton Ok.
MsgBoxResult.Cancel	2	L'utilisateur a appuyé sur le bouton Annuler.
MsgBoxResult.Abort	3	L'utilisateur a appuyé sur le bouton Abandonner.
MsgBoxResult.Retry	4	L'utilisateur a appuyé sur le bouton Réessayer.
MsgBoxResult.Ignore	5	L'utilisateur a appuyé sur le bouton Ignorer.
MsgBoxResult.Yes	6	L'utilisateur a appuyé sur le bouton Oui.
MsgBoxResult.No	7	L'utilisateur a appuyé sur le bouton Non.

Voici un exemple au sein de lequel l'utilisateur est invité à spécifier s'il désire terminer un processus quelconque ou s'il désire continuer à l'aide de la boîte de dialogue ci-contre.

Si l'utilisateur appuie sur le bouton *Oui*, le script s'arrête à l'aide de l'instruction `Exit Do` vue précédemment.


```
Do
 If MsgBox ("Continuer ?", MsgBoxStyle.YesNo) = MsgBoxResult.No Then
        Exit Do
 End If
 MsgBox ("Alors on recommence…", MsgBoxStyle.Information)
Loop

MsgBox ("C'est ici que cela se termine!", MsgBoxStyle.Information)
```

La fonction InputBox

La fonction **`InputBox`** permet de spécifier le texte à afficher au sein de la boîte de dialogue mais également le titre que devra arborer la boîte de dialogue et la valeur par défaut inscrite au sein de la zone de saisie selon le prototype suivant :

```
InputBox ( Message [,Titre] [,Defaut] [,xPos] [,yPos] )
```

Le paramètre optionnel `Defaut` permet de préciser la saisie par défaut affichée à l'utilisateur :

```
Dim strReponse As String
strReponse = InputBox("Entrez votre nom",  , "Votre nom")
```

Ce qui produit le résultat ci-contre. Remarquez que du texte est déjà présent dans la zone de saisie et qu'il ne reste à l'utilisateur qu'à le modifier.

Les paramètres optionnels `xPos` et `yPos` permettent de préciser la position exprimée en pixels à laquelle la boîte de saisie devra s'afficher. La boîte s'affiche au centre de l'écran si aucune valeur pour ces paramètres n'est précisée.

Maintenant, reste à récupérer la réponse de l'utilisateur. La fonction `InputBox` permet de connaître la réponse de l'utilisateur en récupérant la valeur retournée par cette fonction. Dans l'exemple suivant, l'utilisateur est invité à préciser sa date de naissance et cette information est ultérieurement stockée dans la variable `dNaissance`. Ensuite, cette variable peut être testée afin d'en assurer la validité. L'exemple suivant recommence la saisie de la date de naissance si l'information saisie par l'utilisateur est invalide :

```
Dim dNaissance As System.String

'*********************************************************************'
'* S'assure de récupérer une date valide.                            *'
'*********************************************************************'
Do
     dNaissance = InputBox("Entrez votre date de naissance." & ▼
        & vbCrLf & vbCrLf & "Spécifiez la date dans le format " ▼
        & " yyyy-mm-dd")

Loop Until IsDate(dNaissance)
```

Notez que la fonction `InputBox` retourne une chaîne vide "" lorsque l'utilisateur appuie sur le bouton `Annuler` de la boîte de saisie.

Fonctions de traitement des chaînes de caractères

Visual Basic dispose de plusieurs fonctions permettant de manipuler les chaînes de caractères. Cependant, avant de commencer, il peut être important de distinguer les caractères ASCII des caractères Unicode.

Autrefois, les chaînes de caractères étaient constituées de caractères codés sur 8 bits regroupés arbitrairement au sein d'une table dite ASCII. Selon cette table, les lettres majuscules occupaient les positions 65 à 90 (A à Z), les lettre minuscules occupaient les positions 97 à 122 (a à z), les nombres occupaient les positions 48 à 57 (0 à 9) et les autres symboles occupaient diverses autres positions. La valeur ordinale des caractères était et est toujours reconnu internationalement pour les caractères de 0 à 127 dont la table d'association est inscrite ci-dessous. La valeur des caractères supplémentaires étaient alors attribuée selon le système d'exploitation, DOS ou Windows.

Ctrl	Dec	Hex	Char	Code	Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char
^@	0	00		NUL	32	20	sp	64	40	@	96	60	`
^A	1	01	☺	SOH	33	21	!	65	41	A	97	61	a
^B	2	02	☻	SIX	34	22	"	66	42	B	98	62	b
^C	3	03	♥	EIX	35	23	#	67	43	C	99	63	c
^D	4	04	♦	EOI	36	24	$	68	44	D	100	64	d
^E	5	05	♣	ENQ	37	25	%	69	45	E	101	65	e
^F	6	06	♠	ACK	38	26	&	70	46	F	102	66	f
^G	7	07	•	BEL	39	27	'	71	47	G	103	67	g
^H	8	08	◘	BS	40	28	(	72	48	H	104	68	h
^I	9	09	○	HI	41	29	)	73	49	I	105	69	i
^J	10	0A	◙	LF	42	2A	*	74	4A	J	106	6A	j
^K	11	0B	♂	VI	43	2B	+	75	4B	K	107	6B	k
^L	12	0C	♀	FF	44	2C	,	76	4C	L	108	6C	l
^M	13	0D	♪	CR	45	2D	-	77	4D	M	109	6D	m
^N	14	0E	♫	SO	46	2E	.	78	4E	N	110	6E	n
^O	15	0F	☼	SI	47	2F	/	79	4F	O	111	6F	o
^P	16	10	►	SLE	48	30	0	80	50	P	112	70	p
^Q	17	11	◄	CS1	49	31	1	81	51	Q	113	71	q
^R	18	12	↕	DC2	50	32	2	82	52	R	114	72	r
^S	19	13	‼	DC3	51	33	3	83	53	S	115	73	s
^I	20	14	¶	DC4	52	34	4	84	54	T	116	74	t
^U	21	15	§	NAK	53	35	5	85	55	U	117	75	u
^V	22	16	▬	SYN	54	36	6	86	56	V	118	76	v
^W	23	17	↨	EIB	55	37	7	87	57	W	119	77	w
^X	24	18	↑	CAN	56	38	8	88	58	X	120	78	x
^Y	25	19	↓	EM	57	39	9	89	59	Y	121	79	y
^Z	26	1A	→	SIB	58	3A	:	90	5A	Z	122	7A	z
^[	27	1B	←	ESC	59	3B	;	91	5B	[	123	7B	{
^\	28	1C	∟	FS	60	3C	<	92	5C	\	124	7C	¦
^]	29	1D	↔	GS	61	3D	=	93	5D	]	125	7D	}
^^	30	1E	▲	RS	62	3E	>	94	5E	^	126	7E	~
^_	31	1F	▼	US	63	3F	?	95	5F	_	127	7F	⌂

Figure 5.1 – Table ASCII des caractères standards (0 à 127)

Avec les années 90 et l'avènement de l'Internet et l'internationalisation des communications informatiques, la codification ASCII des caractères s'est avérée désuète puisqu'elle ne savait pas intégrer les caractères des différents langages internationaux tels l'arabe et le chinois. Un consortium de fabricants mis alors au point la codification Unicode maintenant reconnue internationalement. Cette nouvelle codification code les caractères sur 16 bits et permet jusqu'à

65536 caractères différents. Visual Basic s'est adapté à cette nouvelle codification et certaines de ses fonctions natives de traitement des chaînes de caractères ont été adaptées afin de traiter cette nouvelle approche de la codification des caractères. Notez donc que l'ensemble des fonctions de traitement des chaînes de caractères de Visual Basic ont été prévues pour traiter des caractères ASCII codés sur 8 bits mais que certaines fonctions prévoient l'utilisation de caractères Unicode codés sur 16 bits. Ces dernières sont explicitement identifiées à l'aide d'un `W` majuscule *(pour Wide Unicode)*.

Fonction	Signification	Exemple
Asc	Retourne la valeur ASCII du premier caractère de la chaîne de caractères spécifiée.	`Dim C As Byte` `C = Asc("A")`
AscW	Retourne la valeur Wide Unicode du premier caractère de la chaîne de caractères spécifiée.	`Dim C As Short` `C = AscW("A")`
Chr	Retourne le caractère correspondant à la valeur ASCII spécifiée.	`Dim C As String` `C = Chr(65)`
ChrW	Retourne le caractère correspondant à la valeur Unicode spécifiée.	`Dim C As String` `C = ChrW(65)`

Filter	Retourne un tableau indexé à zéro contenant les sous-éléments de la chaîne de caractères spécifiée selon un critère de filtre.	`Dim C1() As String= ▼`
		`{"Allo", planète"}`
		`Dim C2() As String`
		`C2 = Filter(C1, ▼`
		`"Allo", True)`

Format	Retourne l'expression spécifiée au sein d'un format spécifié.	`Dim dtA As Date`
		`Dim St As String`
		`dtA = Today()`
		`St = Format(dtA, ▼`
		`"d MMM yyyy")`

FormatCurrency	Retourne en format monétaire l'expression spécifiée.	`Dim stArgent As String`
		`stArgent = ▼`
		`FormatCurrency(12.2)`
		`'Retourne "12.20"`

FormatDateTime	Retourne en format date l'expression spécifiée.	`Dim stDate As String`
		`stDate = ▼`
		`FormatDateTime(Now(),▼`
		`vbLongDate)`

FormatNumber	Retourne en format numérique l'expression spécifiée.	`Dim P, N As Single`
		`N = 5.6`
		`P = FormatNumber(N, 2)`
		`'Retourne 5.60`

FormatPercent	Retourne en pourcentage l'expression spécifiée où 1 vaut 100%.	`Dim st As String`
		`st= FormatPercent(.85)`
		`'Retourne "85%"`

GetChar	Retourne le caractère contenu à la position spécifiée au sein d'une chaîne de caractères.	`Dim St As String`
		`Dim C As Char`
		`St = "Allo"`
		`C = GetChar(St, 3)`
		`'Retourne "o"`

InStr	Retourne un entier spécifiant la position de départ d'une occurrence d'une chaîne de caractères au sein d'une autre chaîne de caractères.	`Dim P As Integer`
		`Dim St As String`
		`St = "Allo"`
		`P = InStr(st, "o")`

InStrRev	Retourne un entier spécifiant la position de départ d'une occurrence d'une chaîne de caractères au sein d'une autre chaîne de caractères à partir de la fin de celle-ci.	`Dim P As Integer`
		`Dim St As String`
		`St = "Allo"`
		`P = InStrRev(st, "o")`

Join	Retourne une chaîne de caractères résultant de la concaténation de deux chaînes spécifiées au sein d'un tableau.	`Dim Ch() As String= ▼`
		`  {"Allo","planète"}`
		`MsgBox(Join(Ch))`

LCase	Retourne une chaîne de caractères copiée d'une chaîne de caractères spécifiée dont les caractères sont convertis en minuscules.	`Dim St1, St2 As String`
		`St1 = "Une phrase"`
		`St2 = LCase(St1)`

Left	Retourne une chaîne de caractères constituée des premiers caractères d'une autre chaîne de caractères selon le nombre de caractères spécifié.	`Dim St1, St2 As String`
		`St1 = "Une phrase"`
		`St2 = Left(St1, 3)`

Len	Retourne le nombre de caractères constituants la chaîne de caractères spécifiée.	`Dim X As Integer` `X = Len("Une phrase")`

LTrim	Retourne une chaîne de caractères constituée d'une copie de la chaîne de caractères spécifiée sans espaces avant le premier caractère imprimable.	`Dim St1, St2 As String`
		`St1 = "   Une phrase "`
		`St2 = LTrim(St1)`
		`'Donne "Une phrase "`

LSet	Retourne la chaîne de caractère spécifiée à la longueur spécifiée en ajoutant des espaces si la longueur de la chaîne source est insuffisante.	`Dim St1, St2 As String`
		`St1 = "Left"`
		`St2 = LSet(St1, 10)`
		`'Retourne "Left     "`

Mid	Retourne la chaîne de caractères contenue à un endroit spécifié au sein d'une autre chaîne de caractères.	`Dim St1, St2 As String`
		`St1 = "Bonjour"`
		`St2 = Mid(St1, 3, 4)`
		`'Donne "njou"`

Replace	Retourne une chaîne de caractères copiée d'une autre au sein de laquelle une expression est remplacée par une expression différente.	`Dim St1, St2 As String`
		`St1 = "Bonjour"`
		`St2 = Replace(St1, ▼`
		`  "o", "?")`
		`'Donne "B?nj?ur"`

Right	Retourne une chaîne de caractères constituée des derniers caractères d'une autre chaîne de caractères selon le nombre de caractères spécifié.	`Dim St1, St2 As String`
		`St1 = "Une phrase"`
		`St2 = Right(St1, 3)`

RSet	Retourne la chaîne de caractère spécifiée à la longueur spécifiée en ajoutant des espaces si la longueur de la chaîne source est insuffisante.	`Dim St1, St2 As String`
		`St1 = "Right"`
		`St2 = RSet(St1, 10)`
		`'Donne "     Right"`

RTrim	Retourne une chaîne de caractères constituée d'une copie de la chaîne de caractères spécifiée sans espaces après le premier caractère imprimable.	`Dim St1, St2 As String`
		`St1 = " Une phrase   "`
		`St2 = RTrim(St1)`
		`'Donne " Une phrase"`

Space	Retourne une chaîne de caractères consistant au nombre d'espaces spécifiées.	`Dim St As String`
		`St = Space(10)`
		`'Donne "          "`

Split	Retourne un tableau uni-dimensionnnel constitué des mots d'une chaîne de caractères spécifiée séparés par le caractère délimiteur spécifié (par défaut le caractère d'espacement)	`Dim St1 As String`
		`Dim St2() As String`
		`St1 = "Look at these!"`
		`St2 = Split(St1)`
		`'Donne ["Look", "at",`
		`"these!"]`

Str	Retourne une chaîne de caractères représentant l'expression spécifiée.	`Dim St As String` `St = Str(Today())`

StrComp	Retourne -1, 0, ou 1 selon le résultat de la comparaison des chaînes de caractères. Les chaînes sont comparées selon un tri alphanumérique.	`Dim St1, St2 As String`
		`Dim C As Integer`
		`St1 = "ABCD"`
		`St2 = "abcd"`
		`C =StrComp(S11, St2)`
		`'C donne 0`

StrConv	Retourne une chaîne de caractères convertie dans le format spécifié.	`Dim St1, St2 As String`
		`St2 = StrConv(St1, ▼`
		`vbProperCase)`

StrDup	Retourne une chaîne de caractères constituée d'une répétition de *n* fois le caractère spécifié.	`Dim St As String`
		`St = StrDup(5, "A")`
		`'Donne "AAAAA"`

StrReverse	Retourne une chaîne de caractères résultant de l'inversion de l'ordre de tous les caractères de la chaîne spécifiée.	`Dim St1, St2 As String`
		`St1 = "Allô"`
		`St2 = StrReverse(St1)`

Trim	Retourne une chaîne de caractères constituée d'une copie de la chaîne de caractères spécifiée sans espaces avant et après le premier caractère imprimable.	`Dim St1, St2 As String`
		`St1 = "  Une phrase  "`
		`St2 = Trim(St1)`
		`'Donne "Une phrase"`

UCase	Retourne une chaîne de caractères copiée d'une chaîne de caractères spécifiée dont les caractères sont convertis en majuscules.	`Dim St1, St2 As String`
		`St1 = "Une phrase"`
		`St2 = UCase(St1)`
		`'Donne "UNE PHRASE"`

Notez que la fonction `Left()` peut entrer en conflit avec la propriété `Left` du formulaire si vous utilisez Visual Basic.NET au sein de Windows Forms. Dans ce cas, il peut être nécessaire de spécifier la librairie à laquelle la fonction appartient, en l'occurrence `Microsoft.VisualBasic` comme le démontre l'exemple suivant :

```
Dim K As Integer
Dim MaPhrase As String = "Allô les programmeurs"

K = Microsoft.VisualBasic.Left(MaPhrase, 3)
```

Fonctions mathématiques

En plus des opérateurs arithmétiques de base, Visual Basic dispose de plusieurs fonctions permettant d'effectuer des calculs mathématiques. À part `Fix()` et `Int()`, les fonctions mathématiques sont déclarées au sein de `System.Math` qui doit être inclus au sein du code désirant utiliser ces fonctions à l'aide de l'instruction suivante :

```
Imports System.Math
```

Examinons ces fonctions mathématiques.

Fonction	Signification	Exemple
Abs	Retourne la valeur absolue de l'expression numérique (sans le signe).	`Abs(-2.5)` 'Retourne 2.5
Atan	Retourne l'arc tangente en radians d'un angle exprimé par le ratio de deux côtés d'un triangle rectangle. Le ratio est la longueur du côté opposé à l'angle divisé par la longueur du côté adjacent à l'angle. Pour convertir les radians en degrés, multipliez les radians par 180/pi.	`Dim PI As Double` `PI = 4 * Atan(1)`
Cos	Retourne le cosinus en radians d'un angle exprimé par le ratio de deux côtés d'un triangle rectangle. Le ratio est la longueur du côté adjacent à l'angle divisé par la longueur de l'hypothénuse. Pour convertir les radians en degrés, multipliez les radians par 180/pi.	`Dim Sect As Double` `Sect = 1 / Cos(1.3)`
Exp	Retourne la valeur de e^x pour un `x` donné.	`Exp(1)` 'Retourne 2.718282
Fix	Retourne la partie entière d'un nombre sans les décimales et sans arrondissement.	`Fix(-5.7)` 'Retourne –5
Int	Retourne la partie entière d'un nombre sans les décimales après arrondissement.	`Int(-5.7)` 'Retourne –6
Log	Retourne le logarithme naturel d'un nombre. Le logarithme naturel, ou népérien, est le logarithme sur la base `e` valant approximativement 2.718282.	`Lg10 = Log(x)/Log(10#)` 'Calcule le logarithme de 10
Round	Retourne un nombre aléatoire situé entre 0 inclusivement et 1 exclusivement.	`(Max - Min + 1) * ▾` `Round() + Min`
Sign	Retourne une valeur indiquant le signe du nombre spécifié. Cette valeur vaut –1 lorsque le nombre est négatif, 1 lorsque le nombre est positif et 0 lorsque le nombre est nul.	`Sign(8)` 'Retourne 1 `Sign(-3)` 'Retourne –1 `Sign(0)` 'Retourne 0
Sin	Retourne le sinus en radians d'un angle exprimé par le ratio de deux côtés d'un triangle rectangle. Le ratio est la longueur du côté opposé à l'angle divisé par la longueur de l'hypothénuse. Pour convertir les radians en degrés, multipliez les radians par 180/pi.	`Dim CoSec As Double` `CoSec = 1 / Sin(1.3)`
Sqrt	Retourne la racine carrée d'un nombre.	`Sqrt(25)` ' Retourne 5
Tan	Retourne la tangente en radians d'un angle exprimé par le ratio de deux côtés d'un triangle rectangle. Le ratio est la longueur du côté opposé à l'angle divisé par la longueur du côté adjacent à l'angle. Pour convertir les radians en degrés, multipliez les radians par 180/pi.	`Dim CoTan As Double` `CoTan = 1 / Tan(1.3)`

Plusieurs fonctions mathématiques ne sont pas intrinsèques à Visual Basic.NET telles le calcul de la sécante ou le calcul d'un sinus hyperbolique. Cependant, ces fonctions mathématiques peuvent être connues à l'aide des fonctions mathématiques intrinsèques à Visual Basic :

Fonction		Équivalent
Cosécante	Csc(x)	`1 / Sin(x)`
Cotangente	Ctan(x)	`1 / Tan(x)`
Sécante	Sec(x)	`1 / Cos(x)`
Sinus inverse	Asin(x)	`Atan(x / Sqrt(-x * x + 1))`
Cosinus inverse	Acos(x)	`Atan(-x / Sqrt(-x * x + 1)) + 2 * Atan(1)`
Sécante inverse	Asec(x)	`2 * Atan(1) - Atan(Sgn(x)/Sqrt(x * x - 1))`
Cosécante inverse	Acsc(x)	`Atan(Sgn(x) / Sqrt(x * x - 1))`
Cotangente inverse	Acot(x)	`2 * Atan(1) - Atan(x)`
Sinus hyperbole	Sinh(x)	`(Exp(x) - Exp(-x)) / 2`
Cosinus hyperbole	Cosh(x)	`(Exp(x) + Exp(-x)) / 2`
Tangente hyperbole	Tanh(x)	`(Exp(x) - Exp(-x)) / (Exp(x) + Exp(-x))`
Sécante hyperbole	Sech(x)	`2 / (Exp(x) + Exp(-x))`
Cosécante hyperbole	Csch(x)	`2 / (Exp(x) - Exp(-x))`
Cotangente hyperbole	Coth(x)	`(Exp(x) + Exp(-x)) / (Exp(x) - Exp(-x))`

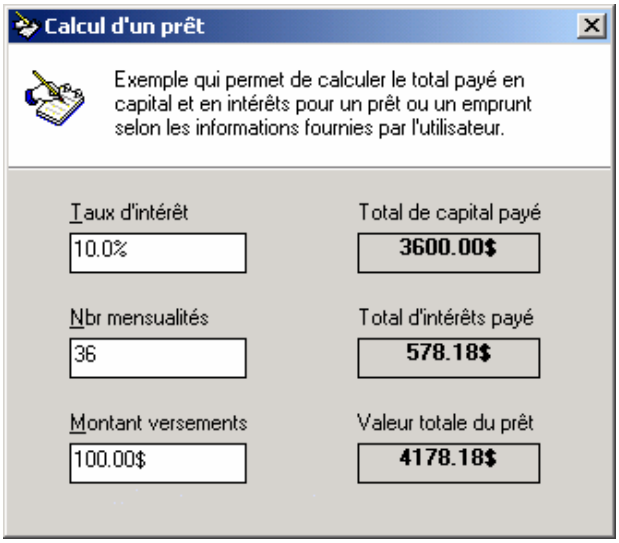
Fonctions financières

Visual Basic dispose de plusieurs fonctions permettant d'effectuer des calculs pour des fins financières comme le calcul de taux d'intérêts et d'annuités. Voici la liste de ces fonctions financières :

Fonction	Signification	Exemple
DDB	Retourne l'amortissement d'un bien pour une période déterminée selon la méthode *« double-declining balance »*.	`Dim X As Double` `X = DDB(cout, val_resid, duree, per)`
FV	Retourne la valeur future d'une annuité selon des versements périodiques et un taux d'intérêts constant.	`Dim X As Double` `X = FV(taux, npmt, pmt, va, type)`
IPmt	Retourne une valeur spécifiant le montant, sur une période donnée, d'une annuité basée sur des versements constants et périodiques et sur un taux d'intérêt fixe	`Dim X As Double` `X = IPmt(APR / 12, Per, TotPmts, -PVal, Fval, PayType)`
IRR	Retourne une valeur spécifiant le taux de rendement interne d'une série de mouvements de trésorerie périodiques.	`Dim X As Double` `X = IRR(Values, Guess) * 100`
MIRR	Retourne une valeur spécifiant le taux de rendement interne modifié d'une série de mouvements de trésorerie périodiques.	`Dim X As Double` `X = MIRR(Values, LoanAPR, InvAPR)`
NPer	Retourne une valeur spécifiant le nombre d'échéances d'une annuité basée sur des versements constants et périodiques et sur un taux d'intérêt fixe.	`Dim X As Double` `X = NPer(APR / 12, -Payment, PVal, FVal, PayType)`
NPV	Retourne une valeur spécifiant la valeur nette actuelle d'un investissement, calculée en fonction d'une série de mouvements de trésorerie périodiques et d'un taux d'escompte.	`Dim X As Double` `X = NPV(RetRate, valeurs)`
PPmt	Retourne une valeur spécifiant le remboursement du capital, pour une échéance donnée, d'une annuité basée sur des versements constants et périodiques et sur un taux d'intérêt fixe.	`Dim X As Double` `X = PPmt(APR / 12, Period, TotPmts, -PVal, FVal, PayType)`
PV	Retourne une valeur spécifiant le montant actuel d'une annuité basée sur des échéances futures constantes et périodiques, et sur un taux d'intérêt fixe.	`Dim X As Double` `X = PV(APR, TotPmts, -YrIncome, FVal, PayType)`
Rate	Retourne une valeur spécifiant le taux d'intérêt par échéance pour une annuité	`Dim X As Double` `X = (Rate(TotPmts, -Payment, PVal, FVal, PayType, Guess) * 12) * 100`
SLN	Retourne une valeur spécifiant l'amortissement linéaire d'un bien sur une période donnée.	`Dim X As Double` `X = SLN(InitCost, SalvageVal, LifeTime)`
SYD	Retourne une valeur spécifiant l'amortissement global d'un bien sur une période donnée.	`Dim X As Double` `X = SYD(InitCost, SlvgVal, LifeTime, DepYear)`

Exemple d'une application financière

L'exemple suivant met en œuvre la conception d'une interface utilisateur à l'aide des contrôles de base de Visual Basic, l'utilisation des fonctions de traitement des chaînes de caractères et l'utilisation de fonctions financières.



L'exemple suivant est constitué d'une application financière permettant de connaître les montants en capital et en intérêts payés sur un emprunt ou sur un prêt selon un taux d'intérêts, un nombre de mensualités et un montant des mensualités précisés par l'utilisateur.

Les informations spécifiées par l'utilisateur seront constamment validées afin que celui-ci ne puisse inscrire que des valeurs valides. De plus, les calculs s'effectueront à mesure que l'utilisateur modifiera les valeurs contenues au sein des boîtes de texte.

Figure 5.2 – Interface utilisateur de l'application financière

La section gauche de l'interface utilisateur est composée de trois boîtes de texte (`TextBox`) et d'autant d'étiquettes (`Label`) permettant d'identifier à l'utilisateur l'utilité de chacune des boîtes de texte.

Composant	Description
TxtTaux	Boîte de texte permettant la saisie du taux d'intérêt.
TxtNbr	Boîte de texte permettant la saisie du nombre de mensualités.
TxtMontant	Boîte de texte permettant la saisie du montant versé mensuellement.

La section droite de l'interface utilisateur est quant à elle composée de six étiquettes (`Label`) regroupés par paire et permettant d'afficher les résultats des calculs de l'application. Les étiquettes au sein de lesquelles les valeurs seront affichées sont identifiées comme suit au sein du code de l'exemple :

Composant	Description
LblCapital	Étiquette affichant le total de capital payé.
LblInterets	Étiquette affichant le total d'intérêts payé.
LblTotal	Étiquette affichant le montant total payé pour ce prêt.

D'abord, initialisez les propriétés `Text` de chacun des composants afin qu'ils possèdent les bonnes valeurs au démarrage de l'application. Ensuite, créez la procédure `Calculer()` comme suit :

```
Imports System.Math
Public Class Form1 Inherits System.Windows.Forms.Form

⊞ Region " Windows Form Designer generated code "

    Private Sub Calculer()
        Dim Taux As Double, Montant As Double
        Dim Total As Double, Nbr As Integer

    End Sub
```

La fonction `Calculer()` effectuera l'ensemble des calculs et des mises à jour de l'affichage à l'utilisateur. Quatre variables sont déclarées afin de stocker les informations saisies par l'utilisateur.

Cependant, ces informations peuvent contenir des caractères indésirables tels les symboles `$` et `%` que nous devrons supprimer afin de rendre les informations utilisables. Nous pouvons utiliser la fonction `Replace()` afin de remplacer les caractères indésirables par des chaînes vides :

```
Nbr = CInt(txtNbr.Text)

Montant = CDbl(Replace(Replace(txtMontant.Text,"$", ""), ".", ","))

Taux = CDbl(Replace(Replace(txtTaux.Text, "%", ""), ".", ","))
```

La fonction financière `FV()` nous permettra ensuite de connaître le montant total d'un prêt :

```
Total = Abs(FV(Taux / 100 / Nbr, Nbr, Montant))
```

Les valeurs trouvées peuvent être affichées à l'utilisateur. La fonction `Format()` nous permettra ici de mettre en forme le texte afin qu'il s'affiche en format monétaire ou en pourcentage :

```
LblTotal.Text = Format(Total, "0.00") & "$"
LblCapital.Text = Format(Montant * Nbr, "0.00") & "$"
LblInterets.Text = Format(Total - (Montant * Nbr), "0.00") & "$"
```

Finalement, on pourrait remettre en forme le texte saisie par l'utilisateur si jamais celui-ci avait supprimé les caractères `$` et `%`.

```
txtTaux.Text = Format(Taux, "0.0") & "%"
txtMontant.Text = Format(Montant, "0.00") & "$"
```

Ensuite, la procédure `Calculer()` est exécutée au chargement du formulaire à l'aide de l'événement `Load()` du formulaire puis à chaque fois que l'utilisateur déplace le focus d'une boîte de texte vers une autre à l'aide de l'événement `Validating()` des boîtes de texte.

```
Private Sub Form1_Load(ByVal sender As System.Object,
  ByVal e As System.EventArgs) Handles MyBase.Load
    Calculer()
End Sub


Private Sub txtTaux_Validating(ByVal sender As Object,
  ByVal e As System.ComponentModel.CancelEventArgs)
  Handles txtTaux.Validating
    Calculer()
End Sub
```

Voici donc le code complet de l'application financière :

```
Imports System.Math
Public Class Form1
    Inherits System.Windows.Forms.Form

⊞ Region " Windows Form Designer generated code "

    Private Sub Calculer()
        Dim Taux As Double, Montant As Double
        Dim Total As Double, Nbr As Integer

        '*******  Assure la validité des valeurs  ************'
        Nbr = CInt(txtNbr.Text)
        Montant = CDbl(Replace(Replace(txtMontant.Text,"$", ""),
            ".", ","))
        Taux = CDbl(Replace(Replace(txtTaux.Text, "%", ""), ".",
            ","))

        '***********  Calcule les valeurs  ******************'
        Total = FV(Taux / 100 / Nbr, Nbr, Montant) * -1

        '******* Affiche les valeurs à l'utilisateur *********'
        LblTotal.Text = Format(Total, "0.00") & "$"
        LblCapital.Text = Format(Montant * Nbr, "0.00") & "$"
        LblInterets.Text = Format(Total - (Montant * Nbr),
            "0.00") & "$"

        txtTaux.Text = Format(Taux, "0.0") & "%"
        txtMontant.Text = Format(Montant, "0.00") & "$"

    End Sub
```

Le code se poursuit sur la page suivante

```
    Private Sub Form1_Load(ByVal sender As System.Object,
      ByVal e As System.EventArgs) Handles MyBase.Load

        Calculer()

    End Sub


    Private Sub txtTaux_Validating(ByVal sender As Object,
      ByVal e As System.ComponentModel.CancelEventArgs)
      Handles txtTaux.Validating

        Calculer()

    End Sub

    Private Sub txtMontant_Validating(ByVal sender As Object,
      ByVal e As System.ComponentModel.CancelEventArgs)
      Handles txtMontant.Validating

        Calculer()

    End Sub


    Private Sub txtNbr_Validating(ByVal sender As Object,
      ByVal e As System.ComponentModel.CancelEventArgs)
      Handles txtNbr.Validating

        Calculer()

    End Sub

End Class
```

Testez votre application de calcul des taux d'intérêts en sélectionnant le menu **Déboguer / Démarrer** ou en appuyant simplement sur la touche **F5** ou **CTRL+F5** selon la configuration de votre clavier.

Fonctions de traitement des dates et de l'heure

Visual Basic dispose d'une pluralité de fonctions permettant d'effectuer des calculs et divers traitements sur les dates. Les fonctions suivantes sont intrinsèques au langage et ne nécessitent l'inclusion d'aucune référence. Examinons ces fonctions de traitement des dates et de l'heure.

Fonction	Signification	Exemple

DateAdd	Retourne une date résultant de l'addition d'un nombre de jour, de mois ou d'années spécifié. La fonction soustrait si la valeur spécifiée est négative.	`Dim dt1, dt2 As Date`
		`dt1 = Today()`
		`dt2 = ▼`
		`DateAdd("d",15, dt1)`

DateDiff	Retourne le nombre de jours, de mois ou d'années espaçant deux dates spécifiées. Les constantes utilisées sont "d" = jours, "m"= mois, "yyyy" = années, "h" = heures, "n" = minutes et "s" = secondes.	`Dim dDiff As Date`
		`dDiff = DateDiff("m", ▼`
		`dNaissance, Today())`

DatePart	Retourne une partie spécifique de la date spécifiée. Peut être le jour, le mois, etc.	`Dim nMois As Integer`
		`Dim dA As Date`
		`dA = Today()`
		`nMois =DatePart(dA, "m")`
		`'Retourne le mois`

DateSerial	Retourne une date en format `Date` à partir d'informations en format sériel.	`MsgBox(DateSerial(1975,▼`
		`11, 3))`
		`'Donne "3 novembre 1975"`

DateString	Retourne la date courante en format `String` selon l'heure système.	`MsgBox(DateString)`

DateValue	Retourne une date en format `Date` à partir d'informations spécifiées en format `String`.	`Dim dtA As Date`
		`dtA = ▼`
		`DateValue("1/31/2000")`

Day	Retourne le numéro du jour contenu au sein d'une expression spécifiée.	`Dim nJour As Integer` `nJour = Day(Today())`
Hour	Retourne les heures contenues au sein d'une expression spécifiée.	`Dim nHr As Integer` `nHr=Hour(TimeOfDay())`
Minute	Retourne les minutes contenues au sein d'une expression spécifiée.	`Dim nMin As Integer` `nMin=Minute(TimeOfDay())`
Month	Retourne le numéro du mois (entre 1 et 12) contenu au sein d'une expression spécifiée.	`Dim nMois As Integer` `nMois = Month(Today())`

MonthName	Retourne le nom du mois correspondant au numéro de mois spécifié.	`Dim nMois As Integer`
		`nMois = Month(Today())`
		`MsgBox "Nous sommes au mois "& MonthName(nMois)`

Now	Retourne la date du jour courant et l'heure courante selon les date et heure systèmes.	`MsgBox Now()`
Second	Retourne les secondes contenues au sein d'une expression spécifiée.	`Dim nSec As Integer` `nSec=Second(TimeOfDay())`
TimeOfDay	Retourne l'heure courante selon l'heure système.	`MsgBox(TimeOfDay)`

TimeSerial	Retourne une date en format `Date` à partir d'informations en format sériel.	`MsgBox(TimeSerial( 23, ▼`
		`75, 0))`
		`'"1/2/0001 12:15:00 AM"`

TimeString	Retourne l'heure courante en format `String` selon l'heure système.	`MsgBox(TimeString)`

TimeValue	Retourne une date en format `Date` à partir d'informations spécifiées en format `String`.	`Dim dtA As Date`
		`dtA = ▼`
		`TimeValue("4:35:17 PM")`

Today	Retourne la date du jour courant selon la date système.	`MsgBox Today()`
WeekDay	Retourne le numéro du jour de la semaine (entre 1 et 7) contenu au sein d'une expression date spécifiée.	`Dim nJour As Integer` `nJour = WeekDay(Today())`

WeekDayName	Retourne le nom du jour de la semaine correspondant au numéro de jour spécifié.	`Dim n As Integer`
		`Dim strJour As String`
		`n = WeekDay(Today())`
		`strJour = WeekDayName(n)`

Year	Retourne l'année contenu au sein d'une expression date spécifiée.	`Dim nAn As Integer` `nAn = Year(Today())`

Exemple d'une application manipulant les dates

L'exemple suivant met en œuvre la conception d'un interface utilisateur à l'aide des contrôles de base de Visual Basic et l'utilisation des fonctions de traitement des dates.

Figure 5.3 – Interface utilisateur de l'application

L'exemple suivant est constitué d'une application permettant de connaître l'âge exact d'une personne selon sa date de naissance.

Les informations spécifiées par l'utilisateur seront saisies par un contrôle `DateTimePicker` assurant que celui-ci ne puisse inscrire que des dates valides. Le calcul s'effectuera lorsque l'utilisateur appuiera sur le bouton *Calculer l'âge*.

Outre les étiquettes permettant d'identifier les contrôles, l'interface utilisateur est composée d'un contrôle `DateTimePicker` et d'un bouton.

Composant	Description
dtNaissance	Contrôle permettant la saisie de la date de naissance.
btnCalculerAge	Bouton permettant de lancer le calcul de l'âge de la personne.

D'abord, initialisez les propriétés de chacun des composants afin qu'ils possèdent les bonnes valeurs au démarrage de l'application. Ensuite, codez l'événement `Click` du bouton comme suit:

```
Private Sub btnCalculerAge_Click(ByVal sender As System.Object,
      ByVal e As System.EventArgs) Handles btnCalculerAge.Click

      Dim dNaissance As Date
      Dim nDiff, nJours As Long
      Dim StrDate As String


      '**** Stocke la date de naissance dans la variable *******'
      dNaissance = dtNaissance.Value


      '*********** Calcule le nombre de mois restants  *********'
      nDiff = DateDiff(DateInterval.Month, dNaissance, Today())


      '********** Calcule le nombre de jours restants  *********'
      nJours = DateDiff("d", DateAdd(DateInterval.Month, nDiff,
            dNaissance), Today())


      '********* Affiche le résultat à l'utilisateur  **********'
      strDate = WeekdayName(Weekday(dNaissance)) & " le "
         & Microsoft.VisualBasic.Day(dNaissance) & " "
         & MonthName(Month(dNaissance)) & " " & Year(dNaissance)


      MsgBox("Si vous êtes né(e) " & StrDate & " vous avez "
         & nDiff \ 12 & " ans, " & nDiff Mod 12 & " mois et "
         & nJours & " jours.", MsgBoxStyle.Information)

End Sub
```

Fonctions de traitement des fichiers et des répertoires

Visual Basic dispose d'une pluralité de fonctions permettant d'effectuer différentes tâches sur les fichiers et les répertoires *(modifier, renommer, créer, supprimer, etc)*. Les fonctions suivantes sont intrinsèques au langage et ne nécessitent l'inclusion d'aucune référence. Examinons ces fonctions de traitement des fichiers et des répertoires.

Fonction	Signification	Exemple
ChDir	Change le répertoire de travail courant.	`ChDir("c:\winnt")`
ChDrive	Change le lecteur de travail courant.	`ChDir("e")`
CurDir	Retourne le chemin du répertoire de travail courant.	`Dim St As String` `St = CurDir()`

Dir	Retourne le nom d'un fichier, d'un répertoire ou d'un lecteur. La fonction retourne une chaîne vide "" si le fichier ou le répertoire n'existe pas.	`Dim StFile As String`
		`StFile = Dir("c:\Winnt\win.ini")`
		`'Retourne "win.ini"`

FileCopy	Copie le fichier spécifié en un nouveau fichier au nom spécifié.	`FileCopy("c:\boot.ini", "d:\boot.ini")`
FileDateTime	Retourne les informations de date de création et de modification d'un fichier.	
FileLen	Retourne le nombre d'octets occupés par le fichier spécifié.	`Dim Lng As Long` `Lng = FileLen("c:\boot.ini")`

GetAttr	Retourne les attributs *(caché, système, etc)* d'un fichier ou d'un répertoire.	`Dim N As Long`
		`N = GetAttr("c:\boot.ini")`
		`If N And vbHidden Then`
		`'Un fichier caché!`
		`End If`

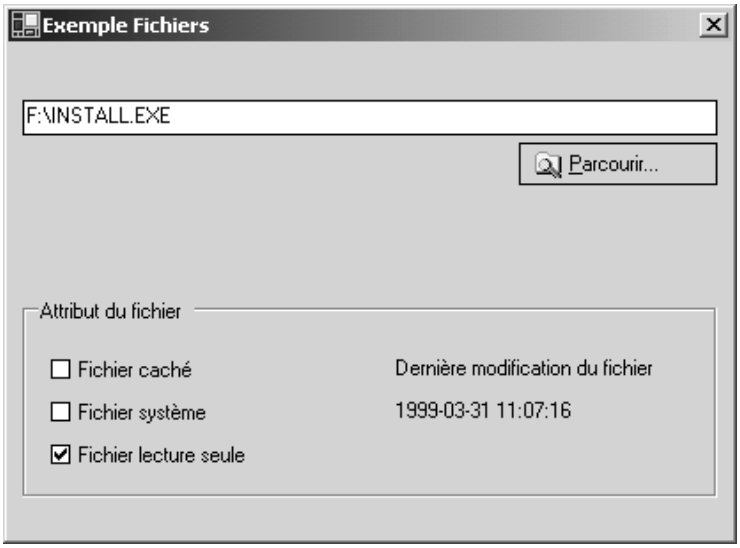
Kill	Procède à la suppression du fichier spécifié.	`Kill ("c:\fichier.txt")`
MkDir	Procède à la création du répertoire spécifié.	`MkDir("c:\dossier")`
Rename	Renomme le répertoire ou le fichier spécifié.	`Rename("c:\dossier1", "c:\dossier2")`
RmDir	Procède à la suppression du répertoire spécifié.	`RmDir("c:\dossier")`
SetAttr	Spécifie les attributs *(caché, système, etc)* d'un fichier ou d'un répertoire.	`SetAttr("c:\boot.ini", vbHidden)` `'Cache le fichier`

Les fonctions `Dir`, `GetAttr` et `SetAttr` utilisent les constantes suivantes afin de connaître les attributs d'un fichier ou d'un dossier. Voyez l'exemple complet suivant afin de mieux comprendre l'utilisation de ces constantes.

Constante	Description
vbNormal	Ne possède aucun attribut particulier.
vbReadOnly	Est en lecture seulement.
vbHidden	Est un fichier ou un dossier caché.
vbSystem	Est un fichier ou un dossier système.
vbDirectory	Est un dossier.
vbArchive	Est un fichier ou un dossier archivé.
vbAlias	Est un alias (raccourci) vers un autre fichier ou dossier.

Exemple d'une application traitant les fichiers

L'exemple suivant met en œuvre la conception d'une interface utilisateur à l'aide des contrôles de base et l'utilisation de fonctions de traitement des fichiers et des répertoires.



L'exemple suivant est constitué d'une application permettant de connaître les attributs d'un fichier ainsi que la date et l'heure de création ou dernière modification de ce fichier.

Un traitement spécial est appliqué si le fichier spécifié par l'utilisateur n'existe pas.

Figure 5.4 – Interface utilisateur de l'application

L'interface utilisateur est composée d'une zone de texte (`TextBox`) et d'un bouton permettant de parcourir les répertoires afin de sélectionner un fichier.

Composant	Description
txtFichier	Boîte de texte permettant la saisie du chemin et nom du fichier.
btnParcourir	Bouton activant une boîte de dialogue commun permettant de parcourir les dossiers à la recherche d'un fichier. Les contrôles de boîte de dialogue commun sera exploré en profondeur au chapitre 6.

Trois cases à cocher permettent d'afficher les attributs du fichier sélectionné par l'utilisateur et un étiquette *(Label)* permet d'afficher la date de création ou de dernière modification du fichier.

Composant	Description
lblDateCreation	Étiquette affichant la date de création ou de dernière modification du fichier.
lblAvertissement	Étiquette affiché à l'utilisateur lorsque le fichier n'existe pas.
chkHidden	Case à cocher affichant si le fichier spécifié est caché.
chkSysten	Case à cocher affichant si le fichier spécifié est un fichier système.
chkReadOnly	Case à cocher affichant si le fichier spécifié est en lecture seule.

Lorsque l'utilisateur spécifie un nom de fichier dans la zone de texte, le contrôle soulève l'événement `TextChanged`. Lorsque l'utilisateur utilise le bouton `Parcourir` pour sélectionner un fichier, le chemin de ce dernier est stocké dans la zone de texte ce qui a pour effet de changer le contenu de celle-ci et de soulever également l'événement `TextChanged`. Nous utiliserons donc cet événement afin de récupérer les attributs du fichier sélectionné par l'utilisateur.

La première chose à faire est de désactiver le groupe des contrôles affichant les attributs du fichier si ce dernier n'existe pas. En effet, le fait de lire les attributs d'un fichier inexistant provoquerait une erreur. Ainsi, nous utiliserons la fonction `Dir()` qui retourne une chaîne vide si le fichier spécifié n'existe pas.

```
Private Sub txtFichier_TextChanged(ByVal sender As Object, ▼
      ByVal e As System.EventArgs) Handles txtFichier.TextChanged


    '*******  Affiche si le fichier existe ou non  ***************'
    LblAvertissement.Visible = Dir(txtFichier.Text) = ""
    GroupBox1.Enabled = Not LblAvertissement.Visible
```

Ensuite, notre code récupérera les attributs du fichier à l'aide de la fonction `GetAttr()` seulement si le fichier spécifié existe. La fonction `GetAttr()` vous retourne l'ensemble des valeurs numériques représentant les attributs du fichier au sein d'une même valeur. Vous pouvez connaître si le fichier possédait un attribut particulier en testant cette valeur à l'aide de l'opérateur `And`. L'exemple suivant permet de savoir si un fichier est en lecture seule :

```
If GetAttr("c:\bootcfg.log") And vbReadOnly Then
      'Le fichier est en lecture seule
End If
```

Finalement, nous utiliserons la fonction `FileDateTime()` afin de connaître la date de création ou de dernière modification du fichier.

```
    '*************  Affiche les attributs du fichier ************'
    Dim nAttributs As Long

    If Dir(txtFichier.Text) <> "" Then
        nAttributs = GetAttr(txtFichier.Text)

        chkHidden.Checked = nAttributs And vbHidden
        chkSystem.Checked = nAttributs And vbSystem
        chkReadOnly.Checked = nAttributs And vbReadOnly


        '**********   Date de création du fichier  ********'
        LblDateCreation.Text = FileDateTime(txtFichier.Text)
    End If

End Sub
```

Voici donc le code complet de l'application de traitement des fichiers :

```
Public Class FrmFichiers
       Inherits System.Windows.Forms.Form

   ⊞  Region " Windows Form Designer generated code "

   Private Sub btnParcourir_Click(ByVal sender As System.Object, ▾
            ByVal e As System.EventArgs) Handles btnParcourir.Click

     '*******  Permet à l'utilisateur de choisir un fichier  ******'
     If OpenDialog.ShowDialog() = DialogResult.OK Then
         txtFichier.Text = OpenDialog.FileName
     End If

   End Sub


   Private Sub txtFichier_TextChanged(ByVal sender As Object, ▾
         ByVal e As System.EventArgs) Handles txtFichier.TextChanged


     '**********  Affiche si le fichier existe ou non  **********'
     LblAvertissement.Visible = Dir(txtFichier.Text) = ""
     GroupBox1.Enabled = Not LblAvertissement.Visible


     '************  Affiche les attributs du fichier  ************'
     Dim nAttributs As Long

     If Dir(txtFichier.Text) <> "" Then
         nAttributs = GetAttr(txtFichier.Text)
         chkHidden.Checked = nAttributs And vbHidden
         chkSystem.Checked = nAttributs And vbSystem
         chkReadOnly.Checked = nAttributs And vbReadOnly

         '**********   Date de création du fichier  ********'
         LblDateCreation.Text = FileDateTime(txtFichier.Text)
     End If

   End Sub

 End Class
```

Testez votre application de traitement des fichiers en sélectionnant le menu `Déboguer → Démarrer` ou en appuyant simplement sur la touche **F5** ou **CTRL+F5** selon la configuration de votre clavier.

Fonctions de lecture et d'écriture des fichiers

Les fonctions suivantes ne permettent pas de traiter différentes informations des fichiers *(date de création, attributs, etc.)* mais permettent plutôt de procéder à l'ouverture de fichiers, et d'y effectuer des opérations de lecture ou d'écriture. La plateforme .NET prévoit des fonctionnalités équivalentes accessibles par l'ensemble des langages et sont regroupées au sein de l'espace nominé `System.IO`.

Fonction	Signification	Exemple

EOF	Retourne `True` lorsque la fin d'un fichier ouvert en mode `Random` ou `Input` est atteinte.	`fch = "c:\MonFichier.txt"`
		`md = OpenMode.Input`
		`FileOpen(1, fch, md)`
		`Do Until EOF(1)`
		`  txt = LineInput(1)`
		`Loop`
		`FileClose(1)`

FileClose	Ferme le fichier spécifié. Si aucun fichier n'est précisé, tous les fichiers couramment ouverts sont fermés.	`fch = "c:\MonFichier.txt"`
		`md = OpenMode.Input`
		`FileOpen(1, fch, md)`
		`FileClose(1)`

FileGet	Lit les données d'un fichier ouvert et les retourne au sein de la variable spécifiée. Valide que lorsque le fichier est ouvert en mode `Random` ou `Binary`.	`Dim S As New String(" ", 9)`
		`fch = "c:\MonFichier.txt"`
		`md = OpenMode.Binary`
		`FileOpen(1, fch, md)`
		`FileGet(1, S)`
		`FileClose(1)`

FileGetObject	Lit les données d'un fichier ouvert et les retourne au sein de la variable spécifiée. Valide que lorsque le fichier est ouvert en mode `Random` ou `Binary`. Cette fonction est plus typée que `FileGet` et permet donc d'éviter toute ambiguïté de types lors de la compilation.	`Dim S As New String(" ", 9)`
		`fch = "c:\MonFichier.txt"`
		`md = OpenMode.Binary`
		`FileOpen(1, fch, md)`
		`FileGetObject(1, S)`
		`FileClose(1)`

FileLen	Retourne les dimensions en octets du fichier spécifié. Le fichier est spécifié à l'aide de son chemin et de son nom.	`fch = "c:\MonFichier.txt"` `Longueur = FileLen(fch)`

FileOpen	Procède à l'ouverture du fichier spécifié dans le mode spécifié. Voyez plus loin les différents modes d'ouverture de fichiers.	`fch = "c:\MonFichier.txt"`
		`md = OpenMode.Input`
		`FileOpen(1, fch, md)`
		`FileClose(1)`

FilePut	Écrit les données de la variable spécifiée au sein d'un fichier ouvert. Valide que lorsque le fichier est ouvert en mode `Random` ou `Binary`.	`Dim S As String = "Allo"`
		`fch = "c:\MonFichier.txt"`
		`md = OpenMode.Binary`
		`FileOpen(1, fch, md)`
		`FilePut(1, S)`
		`FileClose(1)`

FilePutObject	Écrit les données de la variable spécifiée au sein d'un fichier ouvert. Valide que lorsque le fichier est ouvert en mode `Random` ou `Binary`. Cette fonction est plus typée que `FilePut` et permet donc d'éviter toute ambiguïté de types lors de la compilation.	`Dim S As String = "Allo"`
		`fch = "c:\MonFichier.txt"`
		`md = OpenMode.Binary`
		`FileOpen(1, fch, md)`
		`FilePutObject(1, S)`
		`FileClose(1)`

FileWidth	Fixe la largeur d'une ligne d'un fichier ouvert à l'aide de `FileOpen`. Aucune largeur maximale n'est imposée lorsque la valeur spécifiée vaut zéro.	`Dim F As Integer`
		`F = FreeFile()`
		`FileOpen(F, fch, md)`
		`FileWidth(F, 5)`
		`FileClose(F)`

FreeFile	Retourne un identificateur numérique de fichier disponible. Utilisez cette méthode afin d'obtenir un numéro identifiant de manière unique le fichier à ouvrir.	`Dim F As Integer`
		`F = FreeFile()`
		`FileOpen(F, fch, md)`
		`FileClose(F)`

InputString	Lit le nombre de caractères spécifié d'un fichier ouvert et les retourne au sein de la chaîne de caractères spécifiée. Valide que lorsque le fichier est ouvert en mode `Input` ou `Binary`.	`Dim C As Char`
		`FileOpen(1, fch, md)`
		`Do While Not EOF(1)`
		`  C = InputString(1, 1)`
		`Loop`
		`FileClose(1)`

LineInput	Lit une ligne d'un fichier ouvert et la retourne au sein de la chaîne de caractères spécifiée. Valide que lorsque le fichier est ouvert en mode `Input` ou `Binary`.	`Dim Txt As String`
		`FileOpen(1, fch, md)`
		`Do Until EOF(1)`
		`  Txt = LineInput(1)`
		`Loop`
		`FileClose(1)`

Loc	Retourne un entier long contenant la position courant au sein d'un fichier ouvert en lecture ou en écriture.	`Do While Not EOF(1)`
		`  Console.WriteLine(Loc(1))`
		`Loop`

Lock	Verrouille un fichier ouvert afin que les autres processus ne puisse y accéder.	`FileOpen(1, fch, md)`
		`  Lock(1)`
		`  FilePut(1, S)`
		`FileClose(1)`

LOF	Retourne les dimensions en octets du fichier spécifié. Le fichier doit être préalablement ouvert à l'aide de la méthode `FileOpen`.	`fch = "c:\MonFichier.txt"`
		`md = OpenMode.Input`
		`FileOpen(1, fch, md)`
		`  Longueur = LOF(1)`
		`FileClose(1)`

Print	Écrit une ligne dans un fichier ouvert. Valide que lorsque le fichier est ouvert en mode `Input` ou `Binary`.	`Dim Txt As String = "Allô"`
		`FileOpen(1, fch, md)`
		`Print(1, Txt)`
		`FileClose(1)`

PrintLine	Écrit une ligne dans un fichier ouvert. Ajoute automatiquement un saut de ligne après la ligne écrite. Valide que lorsque le fichier est ouvert en mode `Input` ou `Binary`.	`Dim Txt As String = "Allô"`
		`FileOpen(1, fch, md)`
		`PrintLine(1, Txt)`
		`FileClose(1)`

Reset	Ferme l'ensemble des fichiers couramment ouverts. Possède la même utilité que `FileClose` lorsqu'aucun paramètre ne lui est spécifié.	`fch = "c:\MonFichier.txt"`
		`md = OpenMode.Input`
		`FileOpen(1, fch, md)`
		`Reset()`

Seek	Retourne ou détermine la position au sein d'un fichier ouvert en lecture ou en écriture.	`FileOpen(1, fch, md)`
		`  Seek(1, 72)`
		`FileClose(1)`

Unlock	Déverrouille un fichier ouvert et préalablement verrouillé à l'aide de la fonction `Lock` et permet aux autres processus d'y accéder.	`FileOpen(1, fch, md)`
		`  Lock(1)`
		`  FilePut(1, S)`
		`  Unlock(1)`
		`FileClose(1)`

Fonctions d'accès à la base de registre

La base de registre est une base de données permettant de centraliser les différentes configurations des applications, des préférences utilisateurs, des configurations matérielles, etc. Les informations stockées dans la base de registre sont conservées entre les différentes exécutions du système d'exploitation. La base de registre vous permet donc de stocker des informations concernant vos applications et de les récupérer au prochain démarrage de votre application. Visual Basic prévoit quatre fonctions natives d'accès à la base de registres. Ces fonctions permettent de stocker et récupérer des informations au sein de la base de registres au sein de la clé `HKEY_CURRENT_USER\Software\VB and VBA Program Settings`. Les informations que votre application stockera au sein de la base de registre seront conséquemment utilisables séparément par chacun des utilisateurs.

Si vous désirez lire, écrire ou supprimer des informations stockées au sein d'autres clés que celle utilisée nativement par le langage Visual Basic, vous devrez utiliser les fonctions .NET prévues au sein de l'espace nominé *(namespace)* `System.Win32`.

Fonction	Signification	Exemple
DeleteSetting	Provoque la suppression de la clé spécifiée.	`DeleteSetting("MonApp", Cfg")`

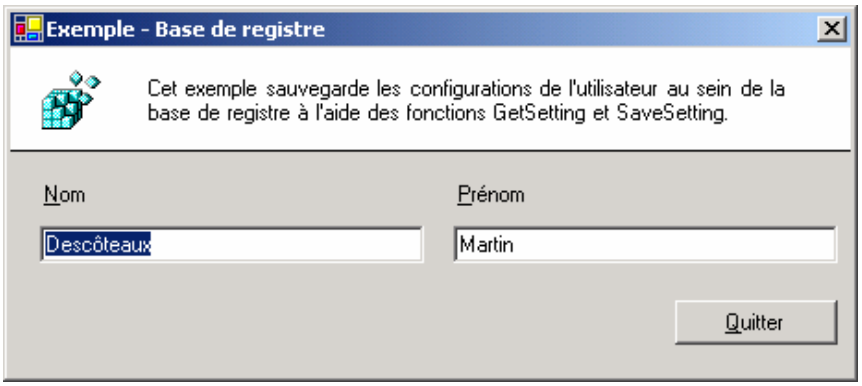
GetSetting	Retourne la valeur contenue au sein de la base de registre pour la clé spécifiée.	`Dim St As String`
		`St = GetSetting("MonApp", ▼`
		`"Cfg", "Valeur", "0")`

GetAllSettings	Retourne la valeur contenue au sein de la base de registre pour la clé spécifiée.	`Dim St(,) As String`
		`St =GetAllSettings("MonApp", ▼`
		`"Cfg")`

SaveSetting	Spécifie la valeur à inscrire au sein de la base de registre pour la clé spécifiée. La clé est créée si celle-ci n'existe pas.	`SaveSetting("MonApp", "Cfg",▼` `"Valeur", "0")`

Exemple d'une application utilisant la base de registre

L'exemple suivant met en œuvre la conception d'un interface utilisateur à l'aide des contrôles de base de Visual Basic, l'utilisation des événements du formulaire et l'utilisation de fonctions d'accès à la base de registre.



L'exemple suivant est constitué d'une application permettant de conserver les préférences de l'utilisateur au sein de la base de registre pour d'ultérieures utilisations.

Figure 5.5 – Interface utilisateur de l'application

Les nom, prénom et dimensions du formulaire seront conservés entre chacune des utilisations du logiciel afin d'être rappelés au démarrage de celui-ci.

L'interface utilisateur est composée de deux zones de texte (`TextBox`) et d'autant d'étiquettes (`Label`) permettant d'identifier à l'utilisateur l'utilité de chacune des boîtes de texte.

Composant	Description
TxtNom	Boîte de texte permettant la saisie du nom de l'utilisateur.
TxtPrenom	Boîte de texte permettant la saisie du prénom de l'utilisateur.

Finalement, un bouton permet à l'utilisateur de quitter l'application.

Composant	Description
BtnQuitter	Bouton permettant de mettre fin à l'application.

D'abord, localisez l'événement **`Closing`** du formulaire, événement soulevé lorsque le formulaire est fermé. Ainsi, lorsque l'utilisateur quittera l'application, notre code en profitera pour enregistrer les préférences de l'utilisateur au sein de la base de registre. Inscrivez votre code avant celui automatiquement généré par l'assistant de Visual Studio.NET comme suit :

```
Private Sub FrmRegistre_Closing( ByVal sender As Object, ▼
            ByVal e As System.ComponentModel.CancelEventArgs) ▼
                    Handles MyBase.ClosingOverloads

   '**** Inscrit les informations dans la base de registre  *****'
   SaveSetting("ExempleRegistre.NET", "Config", "Nom", ▼
       txtNom.Text)
   SaveSetting("ExempleRegistre.NET", "Config", "Prenom", ▼
       txtPrenom.Text)
   SaveSetting("ExempleRegistre.NET", "Config", "Hauteur", ▼
       Me.Height)
   SaveSetting("ExempleRegistre.NET", "Config", "Largeur", ▼
       Me.Width)

End Sub
```

Maintenant , il suffit de récupérer les enregistrements au démarrage de l'application. La fonction `GetSetting` nous permet de lire au sein de la base de registre. Par contre, il est important de porter attention de lire les mêmes clés que celles qui ont été précédemment inscrites.

```
Private Sub FrmRegistre_Load( ByVal sender As System.Object, ▼
            ByVal e As System.EventArgs) Handles MyBase.Load

   '****** Charge les informations de la base de registre  *****'
   txtNom.Text = GetSetting("ExempleRegistre.NET", "Config", ▼
       "Nom", "")
   txtPrenom.Text = GetSetting("ExempleRegistre.NET", "Config", ▼
       "Prenom", "")
   Me.Width = CLng(GetSetting("ExempleRegistre.NET", "Config", ▼
       "Largeur", Me.Width))
   Me.Height = CLng(GetSetting("ExempleRegistre.NET", "Config", ▼
       "Hauteur", Me.Height))
End Sub
```

Finalement, nous pouvons rendre notre application plus fonctionnelle en permettant à l'utilisateur de quitter l'application en cliquant sur le bouton `Quitter`. L'objet `Form` possède la méthode `Dispose()` qui permet de forcer la fermeture du formulaire. Lorsque l'ensemble des formulaires sont fermés, l'application est automatiquement stoppée. Vous pouvez également utiliser la méthode `Application.Exit` afin de quitter l'application si les permissions de l'utilisateur en cours le permettent.

```
Private Sub btnQuitter_Click(ByVal sender As System.Object, ▼
          ByVal e As System.EventArgs) Handles btnQuitter.Click

    Application.Exit()

End Sub
```

Testez votre application d'accès à la base de registres en sélectionnant le menu `Déboguer → Démarrer` ou en appuyant simplement sur la touche **F5** ou **CTRL+F5** selon la configuration de votre clavier.