

Classes de manipulation de dates et d'heures

La structure *DateTime*

La structure *DateTime* permet d'effectuer diverses opérations sur les dates : déterminer le jour de la semaine, calculer une date à x jours, déterminer si une date est antérieure ou postérieure à une autre, etc. Un objet *DateTime* peut contenir une date comprise entre le premier janvier de l'an un à midi et le 31 décembre 9999. Il comprend à la fois une date et une heure avec accès à la milliseconde. On parle de classe *DateTime* bien qu'il s'agisse plus précisément d'une structure.

Nous ne considérerons ici que le calendrier grégorien, celui qui est en vigueur dans nos contrées. L'espace de noms *System.Globalization* contient cependant (outre la classe *GregorianCalendar* automatiquement sélectionnée) les classes *HebrewCalendar*, *HijriCalendar*, *JapaneseCalendar*, *JulianCalendar*, *KoreanCalendar*, *TaiwanCalendar* et *ThaiBuddistCalendar* qui permettent d'utiliser d'autres systèmes de datation.

Les différents constructeurs de la classe *DateTime* sont :

Constructeurs de la classe <i>DateTime</i>	
<i>DateTime</i> ← <i>ValueType</i> ← <i>Object</i>	
using System;	
<i>DateTime</i> (int année, int mois, int jour);	Crée un objet <i>DateTime</i> en spécifiant l'année, le mois et le jour. Pour le 14 juillet 1789, on écrit : <i>DateTime</i> dt=new <i>DateTime</i> (1789, 7, 14);
<i>DateTime</i> (int a, int m, int j, int heure, int minutes, int secondes);	Crée un objet <i>DateTime</i> en spécifiant une date (année, mois et jour) ainsi que l'heure, les minutes et les secondes.
<i>DateTime</i> (int a, int m, int j, int heure, int minutes, int secondes, int milliSec);	Même chose mais avec les millisecondes en plus.
<i>DateTime</i> (long);	Crée un objet <i>DateTime</i> en spécifiant le nombre de ticks (intervalles de cent nanosecondes) depuis le premier janvier de l'an un à midi.

Tous ces constructeurs présentent une autre forme qui permet de spécifier un calendrier en dernier et supplémentaire argument (il s'agit d'un objet d'une des classes mentionnées plus haut).

Propriétés de la classe <i>DateTime</i>		
Date	<i>DateTime</i>	Donne la date, avec la partie heure initialisée à minuit. Il s'agit d'une propriété en lecture uniquement. Cette propriété permet d'isoler la partie date (au sens usuel du terme) d'un objet <i>DateTime</i> .
Day	int	Jour d'une date (valeur entre 1 et 31).
DayOfWeek	<i>DayOfWeek</i>	Jour de la semaine. <i>DayOfWeek</i> renvoie une des valeurs de l'énumération <i>DayOfWeek</i> : Sunday (dimanche), Monday, Tuesday, Wednesday, Thursday, Friday et Saturday (samedi). À ces mnémoniques correspondent les valeurs 0 pour dimanche, 1 pour lundi et 6 pour samedi. <i>DateTime</i> dt = new <i>DateTime</i> (1789, 7, 14); switch (dt.DayOfWeek) { case <i>DayOfWeek</i> .Tuesday :; break; }
		Mais on peut aussi écrire : n = (int)dt.DayOfWeek; n contient 2 : c'était un mardi.

Propriétés de la classe DateTime (suite)

DayOfYear	int	Jour de l'année (valeur comprise entre 1 et 366).
Hour	int	Partie « heure » de la date (valeur entre 0 et 23).
Millisecond	int	Partie « millisecondes » de la date (valeur entre 0 et 999).
Minute	int	Partie « minute » de la date (valeur entre 0 et 59).
Month	int	Mois (valeur entre 1 et 12).
Now	DateTime	Propriété statique qui donne la date du jour (y compris les heures, minutes, etc.) au moment d'exécuter la propriété : <code>DateTime dt = DateTime.Now;</code>
Second	int	Partie « seconde » de la date (valeur entre 0 et 59).
Ticks	long	Nombre d'intervalles de cent nanosecondes qui se sont écoulés depuis le premier janvier de l'an un à midi.
TimeOfDay	TimeSpan	Durée depuis minuit. La classe TimeSpan est présentée plus loin dans ce chapitre.
Today	DateTime	Propriété statique qui donne la date du jour (mais avec l'heure initialisée à zéro heure). Semblable à Now mais ne tient pas compte de l'heure.
UtcNow	DateTime	Semblable à Now (il s'agit donc d'une propriété statique) mais donne la date et l'heure en temps universel (temps universel coordonné, encore souvent appelé « temps de Greenwich »).
Year	int	Partie « année » de la date.

Méthodes de la classe DateTime

DateTime Add(TimeSpan interv);	Renvoie la date correspondant à la date de l'objet sur lequel porte l'opération et à laquelle on ajoute l'intervalle de temps interv. L'opérateur <code>+=</code> peut remplacer cette fonction. Pour obtenir la date dans cent jours : <code>DateTime dt = DateTime.Today; dt += new TimeSpan(100, 0, 0, 0);</code>
DateTime AddDays(double n);	Renvoie la date dans n jours (n pouvant être négatif et la partie fractionnaire indiquant une partie de jour, par exemple 0.25 pour six heures) : <code>dt = dt.AddDays(100);</code> On trouve aussi, sous le même format : AddYears, AddMonths, AddHours, AddMinutes, AddSeconds et AddMilliseconds.
int Compare(DateTime t1, DateTime t2);	Méthode statique qui compare deux dates. Compare renvoie : 0 si les deux dates sont les mêmes ; 1 Si t1 est antérieur à t2 ; -1 Si t1 est postérieur à t2. Les opérateurs <code>==</code> , <code>!=</code> , <code><</code> , <code><=</code> , <code>></code> et <code>>=</code> peuvent être utilisés au lieu de Compare et de CompareTo pour comparer deux dates.

<code>int CompareTo(DateTime t);</code>	Compare la date de l'objet sur lequel porte la fonction avec t. CompareTo renvoie :
	zéro si les deux dates sont les mêmes ;
	une valeur positive si la date de l'objet est postérieure à t ;
	une valeur négative si la date de l'objet est antérieure à t.
	Les opérateurs ==, !=, <, <=, > et >= peuvent être utilisés à la place des fonctions de comparaison.
<code>int DaysInMonth(int année, int mois);</code>	Méthode statique qui renvoie le nombre de jours du mois spécifié en argument : <code>n = DateTime.DaysInMonth(1900, 2);</code> n contient 28 (1900 n'était pas une année bissextile puisque sont bissextiles les années multiples de quatre, sauf les années centenaires bien que les années multiples de 400 le soient).
<code>string ToString(string format);</code>	Met en format une date. Voir les formats et exemples plus loin dans ce chapitre.
<code>DateTime Parse(string s);</code>	Méthode statique qui renvoie une date à partir d'une chaîne de caractères. Parse génère : <ul style="list-style-type: none"> - l'exception FormatException si s ne contient pas une date valide ; - l'exception ArgumentException si s est une chaîne nulle. <code>string s = "14/7/1789";</code> <code>DateTime dt = DateTime.Parse(s);</code> Il est préférable de placer cette dernière instruction dans un try/catch pour intercepter les erreurs sur date. Par défaut, Parse tient compte des caractéristiques régionales pour la représentation des dates. Nous montrerons plus loin comment spécifier une autre représentation de date.
<code>bool IsLeapYear(int année);</code>	Méthode statique qui renvoie true si l'année est bissextile.
<code>TimeSpan Subtract(DateTime dt);</code>	Soustrait deux dates : celle sur laquelle porte la fonction et dt. L'opérateur - peut être utilisé au lieu de la fonction Subtract.
<code>string ToShortTimeString();</code>	Renvoie l'heure sous forme HH:MM. Par exemple : <code>DateTime dt = DateTime.Now;</code> <code>string s = dt.ToShortTimeString();</code>
<code>string ToLongTimeString();</code>	Renvoie l'heure sous forme HH:MM:SS.
<code>string ToShortDateString();</code>	Renvoie la date sous forme jj/mm/aaaa. Par exemple : <code>DateTime dt =</code> <code>new DateTime(1789, 7, 14);</code> <code>string s = dt.ToShortTimeString();</code> s contient 14/7/1789.

Méthodes de la classe DateTime (suite)

`string ToLongDateString();` Renvoie la date en format long. Par exemple (sans oublier que le format dépend des paramètres nationaux de configuration) :

mardi 14 juillet 1789

`DateTime ToUniversalTime();` Convertit l'heure (sur laquelle porte la fonction) en temps universel (autrefois appelé *Greenwich Mean Time*).

Pour copier dans `s` l'heure UTC :

`s = DateTime.UtcNow.Hour + ":" + DateTime.UtcNow.Minute;`

`DateTime.UtcNow.Hour` aurait pu être remplacé par `DateTime.Now.ToUniversalTime().Hour`.

Les informations sur le fuseau horaire sont données par la classe `TimeZone` :

Classe TimeZone		
<code>TimeZone ← Object</code>		
Propriétés de la classe TimeZone		
<code>CurrentTimeZone</code>	<code>TimeZone</code>	Propriété statique qui représente le fuseau horaire local.
<code>DaylightName</code>	<code>str</code>	Nom du fuseau horaire appliquant l'heure d'été : <code>string s = TimeZone.CurrentTimeZone.DaylightName;</code> s contient Paris, Madrid (heure d'été)
<code>StandardName</code>	<code>str</code>	Nom du fuseau horaire (par exemple, Paris, Madrid).
Méthodes de la classe TimeZone		
<code>DaylightTime GetDaylightChanges(int année);</code>		Renvoie la période d'application de l'heure d'été pour l'année passée en argument. Voir exemple ci-dessous.
<code> TimeSpan GetUtcOffset(DateTime);</code>		Renvoie le décalage par rapport au méridien de Greenwich (ce décalage dépend de la date en raison de l'heure d'été).
<code>bool IsDaylightSavingTime(DateTime);</code>		Indique si la date passée en argument correspond à une heure d'été.

Pour déterminer le début de l'heure d'été en 2010, on écrit :

```
using System.Globalization;
...
DateTime dt = TimeZone.CurrentTimeZone.GetDaylightChanges(2010).Start;
```

La propriété `End` de la classe `DaylightTime` donne la date de fin de l'heure d'été. `dt` étant de type `DateTime`, on peut lui appliquer des fonctions comme `ToLongDateString` pour convertir l'heure en une chaîne de caractères.

La structure TimeSpan

La structure `TimeSpan` intervient dans les calculs sur dates (plus précisément dans les intervalles entre dates). Ici aussi, nous parlons de classe alors qu'il s'agit plus précisément d'une structure.

Structure TimeSpan

TimeSpan ← ValueType ← Object

Propriétés de la structure TimeSpan

Days	int	Nombre de jours de l'intervalle de temps (arrondi vers le bas). Pour obtenir l'intervalle, il faut encore y ajouter Hours, Minutes, Seconds et Milliseconds.
TotalDays	double	Nombre de jours de l'intervalle (avec partie fractionnaire pour le nombre d'heures).
Hours	int	Nombre d'heures de l'intervalle (valeur comprise entre 0 et 23).
TotalHours	double	Même chose mais sans limite et avec partie fractionnaire.
Minutes	int	Nombre de minutes (valeur comprise entre 0 et 59).
TotalMinutes	double	Nombre total de minutes.
Seconds	int	Nombre de secondes, entre 0 et 59.
TotalSeconds	double	Nombre total de secondes.
Milliseconds		Nombres de millisecondes (de 0 à 999).
TotalMilliseconds	int	Nombre total de millisecondes de l'intervalle.
Ticks	long	Nombre d'intervalles de cent nanosecondes.

Constructeurs de la structure TimeSpan

TimeSpan(long);	Intervalle spécifié en nombre de ticks (cent nanosecondes).
TimeSpan(int jours, int heures, int minutes, int secondes);	Intervalle spécifié en jours, heures, minutes et secondes.
TimeSpan(int heures, int minutes, int secondes);	Intervalle spécifié en heures, minutes et secondes.

Méthodes de la structure TimeSpan

TimeSpan Add(TimeSpan);	Ajoute un intervalle de temps à un intervalle de temps. L'opérateur + peut être utilisé pour additionner deux TimeSpan, ce qui donne un TimeSpan.
TimeSpan Subtract(TimeSpan);	Soustrait l'intervalle de temps passé en argument de celui sur lequel porte l'opération. L'opérateur - peut être utilisé pour soustraire deux TimeSpan, ce qui donne un TimeSpan.
TimeSpan Duration();	Renvoie la durée, en valeur absolue, d'un intervalle de temps.
TimeSpan FromDays(double val);	Méthode statique qui renvoie un objet TimeSpan à partir d'un nombre fractionnaire de jours. Des méthodes semblables sont FromHours, FromMinutes, FromSeconds et FromMilliseconds.
TimeSpan Parse(string s);	Méthode statique qui renvoie un objet TimeSpan à partir d'une chaîne de caractères. Une exception FormatException est générée si la chaîne n'est pas valide. string s="1:30"; TimeSpan ts = TimeSpan.Parse(s); ts.TotalMinutes vaut 90.

Les opérateurs suivants ont été redéfinis pour TimeSpan : +, -, ==, !=, <, <=, > et >=.

Pour calculer le nombre de jours depuis le début du troisième millénaire (premier janvier de l'an 2001) :

```
DateTime dj=DateTime.Today, d=new DateTime(2001, 1, 1);
TimeSpan ts = dj - d;
int n = ts.Days;
```

Mise en format de date

Les spécificateurs ci-après permettent de mettre une date dans un format déterminé. On suppose que les instructions ci-dessous sont exécutées sur une machine aux caractéristiques de la France (format jj/mm/aa de date, ce qui n'est pas universellement répandu).

Formats généraux de dates

d Date au format court. Par exemple :

```
DateTime dt = new DateTime(1789, 7, 14);

string s = dt.ToString("d"); // s contient 14/7/1789
```

D Date au format long mardi 14 juillet 1789

f Date plus heure : mardi 14 juillet 1789 08:05

F Même chose mais les secondes sont affichées.

g Format court avec heure : 14/7/1789 08:05

G Même chose mais avec les secondes.

M Jour plus mois en clair (même effet avec m) : 14 juillet

R Date GMT en anglais (R ou r) : Tue, 14 Jul 1789 08:05:00 GMT

s Date convenant pour tris : 1789-07-14T08:05:00

t Heure : 08:05

T Heure avec secondes : 08:05:00

u Semblable à s mais date en temps universel (heure GMT donc, ce qui peut avoir une répercussion sur la date).

U Format long de date et d'heure mais en temps universel.

y Mois plus année (même effet que Y) : juillet 1789

Si les indicateurs précédents permettent de spécifier un format général de date, il est possible de personnaliser la représentation de la date à l'aide des indicateurs ci-après. Nous donnerons de nombreux exemples par la suite.

Formats personnalisés de date

:

Séparateur des heures dans une date (celui-ci, qui est : par défaut, est en fait un paramètre de Windows).

/ Séparateur de date (/ par défaut) tel que défini dans les paramètres nationaux de Windows.

d Jour sous forme d'un entier entre 1 et 31.

dd Jour, entre 01 et 31.

ddd	Abréviation du jour (sam. pour samedi par exemple).
dddd	Nom complet du jour (dans la langue d'installation de Windows).
M	Mois, de 1 à 12.
MM	Mois, de 01 à 12.
MMM	Abréviation du mois, par exemple jan. pour janvier.
MMMM	Nom complet du mois.
y	Année de 1 à 99.
yy	Année, de 00 à 99.
yyyy	Année, de 1 à 9999.
h	Heure, de 0 à 11.
hh	Heure, de 00 à 11.
H	Heure, de 0 à 23.
HH	Heure, de 00 à 23
m	Minutes, de 0 à 59.
mm	Minutes, de 00 à 59.
s	Secondes, de 0 à 59.
ss	Secondes, de 00 à 59.

Avec la représentation personnalisée, les autres caractères sont repris tels quels. Les caractères réservés peuvent être insérés à condition de les placer entre ' (single quote).

Pour illustrer le sujet, analysons quelques instructions de mise en format (remplacer xyz par une chaîne de caractères et analyser le résultat s). Les trois premiers exemples sont des formats généraux de date.

```
// quatorze juillet 1789 à huit heures et cinq minutes
DateTime dt = new DateTime(1789, 7, 14, 8, 5, 0);
string s = dt.ToString(xyz);
```

Exemples de mise en format de dates	
xyz	s
"d"	14/7/89
"D"	Mardi 14 juillet 1789
"s"	1789-7-14T08:05
"d-MM-yyyy"	14-07-1789
"dddd, le d MMMM yyyy"	mardi, le 14 juillet 1789
"le dd/MM/yyyy à H:mm"	le 14/07/1789 à 8:05
"le d/M"	le 14/7
"H heures mm"	8 8heure0 05 (confusion sur le h et le s de heures)
"H 'heures' mm"	8 heures 05
"H 'H' mm"	8 H 05

La représentation d'une date dépend des paramètres régionaux. Ainsi, nous (c'est-à-dire fr-FR) affichons la date 14/7/1789, alors que les Américains (en-US) afficheraient 7/14/1789. Les Anglais (en-GB) afficheraient également 14/7/1789.

Il est possible de spécifier en argument supplémentaire de `ToString` une culture spécifique et cette modification de culture est alors limitée au `ToString` (encore faut-il que la culture en question ait été installée sous Windows, ce qui est automatiquement le cas pour "en-US" qui correspond à la culture américaine) :

```
using System.Globalization;  
.....  
string s = dt.ToString(); // s contient 14/7/1789 08:05:00  
s = dt.ToString(new CultureInfo("en-US")); // s contient 7/14/1789 8:05:00 AM  
s = dt.ToString("D"); // s contient mardi 14 juillet 1789  
s = dt.ToString("D", new CultureInfo("en-US")); // s contient Tuesday, July 14, 1789
```

Il en va de même pour `Parse`. Écrire :

```
string s = "14/7/1789";  
DateTime dt = DateTime.Parse(s);
```

est correct si la culture est française mais le programme qui fonctionne correctement chez nous donnera une erreur grave s'il est exécuté aux États-Unis ou dans la plupart des régions du monde.

`Parse` sans argument est préférable si la chaîne de caractères contenant la date provient d'une zone d'édition : l'utilisateur saisi la date dans le format de sa culture et `Parse` s'adapte automatiquement à cette culture.

Si une date est codée « en dur » dans le programme et que celui-ci a une diffusion internationale, il est préférable d'écrire :

```
string s = "7/14/1789";  
DateTime dt = DateTime.Parse(s, new CultureInfo("en-US"));
```

Windows connaît la représentation des jours de la semaine et celle des mois, y compris les représentations abrégées. Il est possible de modifier cette représentation en créant un objet `CultureInfo` et en modifiant les propriétés `DayNames`, `MonthNames`, `AbbreviatedDayNames` ou `AbbreviatedMonthNames` (toutes de type `string[]`) de sa propriété `DateTimeFormat` :

```
using System.Globalization;  
.....  
CultureInfo ci = new CultureInfo("fr-FR");  
string[] dn = {"Di", "Lu", "Ma", "Me", "Je", "Ve", "Sa"};  
ci.DateTimeFormat.DayNames = dn;  
s = dt.ToString("D", ci);
```

Pour obtenir ces informations (noms de mois) pour la « culture » courante :

```
string[] ts = CultureInfo.CurrentCulture.DateTimeFormat.MonthNames;
```

Le tableau de mois correspondant à MonthNames comprend treize entrées, la dernière étant vide.
Le tableau correspondant à DayNames comprend sept cellules, avec Dimanche dans la première.

4.4 Mesure d'intervalles de temps

Pour mesurer un intervalle de temps, il est possible d'écrire :

```
DateTime t1 = DateTime.Now;           // temps de départ  
.....                           // exécuter des instructions  
DateTime t2 = DateTime.Now;           // temps d'arrivée  
TimeSpan ts = t2 - t1;
```

ts.TotalMilliseconds donne alors la durée, exprimée en millisecondes, entre les deux événements.

Depuis la version 2, .NET a introduit la classe Stopwatch (de l'espace de noms System.Diagnostics) afin de mesurer des temps avec grande précision.

Classe Stopwatch		
Stopwatch ← ValueType ← Object		
using System.Diagnostics;		
Propriétés de la classe Stopwatch		
Elapsed	TimeSpan	Durée de type TimeSpan entre Start et Stop.
ElapsedMilliseconds	long	Durée, en millisecondes.
Méthodes de la classe Stopwatch		
Start();		Démarre le chronomètre.
Stop();		Arrête le chronomètre.
bool IsRunning();		Indique si le chronomètre est actif.

On écrira par exemple :

```
using System.Diagnostics;  
Stopwatch sw = new Stopwatch();  
sw.Start();                      // démarrer le chronomètre  
.....  
sw.Stop();                        // l'arrêter
```

La propriété IsHighResolution, de type bool, indique si la résolution haute a pu être utilisée pour mesurer des temps. sw.ElapsedMilliseconds donne alors le nombre de millisecondes qui se sont écoulées entre Start et Stop.