

# Tutoriel : Apprendre à déboguer le code C# avec Visual Studio

27/11/2018 • 11 minutes de lecture •  

## Dans cet article

[Prérequis](#)

[Créer un projet](#)

[Démarrez le débogueur !](#)

[Définir un point d'arrêt et démarrer le débogueur](#)

[Parcourir le code dans le débogueur avec les commandes d'exécution pas à pas](#)

[Parcourir le code avec Exécuter jusqu'au clic](#)

[Modifier le code et continuer le débogage](#)

[Pas à pas sortant](#)

[Redémarrer rapidement votre application](#)

[Inspecter des variables avec des bulles d'informations \(datatips\)](#)

[Inspecter des variables avec les Fenêtres Automatique et Variables locales](#)

[Définir un espion](#)

[Examiner la pile des appels](#)

[Changer le flux d'exécution](#)

[Étapes suivantes](#)

Cet article présente les fonctionnalités du débogueur Visual Studio dans une procédure pas à pas. Pour un tour d'horizon plus général des fonctionnalités du débogueur, voir [Présentation du débogueur](#). Quand vous *débuguez votre application*, cela signifie généralement que vous exécutez votre application en y ayant attaché le débogueur. Quand vous faites cela, le débogueur fournit de nombreuses façons de voir ce que fait votre code pendant qu'il s'exécute. Vous pouvez parcourir votre code pas à pas et examiner les valeurs stockées dans les variables, vous pouvez définir des espions sur des variables pour voir quand les valeurs changent, vous pouvez examiner le chemin d'exécution de votre code, voir si une branche de code s'exécute, etc. Si c'est la première fois que vous essayez de déboguer du code, vous pouvez lire [Débogage pour grands débutants](#) avant de poursuivre cet article.

Bien que l'application de démonstration soit écrite en C#, la plupart des fonctionnalités sont applicables à C++, Visual Basic, F#, Python, JavaScript et d'autres langages pris en charge par Visual Studio (F# ne prend pas en charge Modifier et continuer. F# et JavaScript ne prennent pas en charge la fenêtre **Automatique**). Les captures d'écran sont en C#.

Dans ce didacticiel, vous allez effectuer les actions suivantes :

- ✓ Démarrer le débogueur et atteindre des points d'arrêt
- ✓ Découvrir les commandes permettant de parcourir le code pas à pas dans le débogueur
- ✓ Inspecter des variables dans des bulles d'informations et dans les fenêtres du débogueur
- ✓ Examiner la pile des appels

## Prérequis

Visual Studio 2019 et la charge de travail **Développement .NET Desktop** doivent être installés.

Si vous n'avez pas encore installé Visual Studio, accédez à la page [Téléchargements Visual Studio](#) pour l'installer gratuitement.

Si vous devez installer la charge de travail, mais que vous avez déjà installé Visual Studio, cliquez sur **Outils > Obtenir les outils et fonctionnalités...** , qui ouvre Visual Studio Installer. Visual Studio Installer est lancé. Choisissez la charge de travail **Développement .NET Desktop**, puis choisissez **Modifier**.

## Créer un projet

### 1. Ouvrez Visual Studio.

Appuyez sur **Échap** pour fermer la fenêtre de démarrage. Tapez **Ctrl+Q** pour ouvrir la zone de recherche, tapez **console**, choisissez **Modèles**, puis choisissez **Créer un projet d'application console (.NET Framework)** . Dans la boîte de dialogue qui s'affiche, tapez un nom comme **get-started-debugging**, puis choisissez **Créer**.

Si vous ne voyez pas le modèle de projet **Application console (.NET Framework)** , accédez à **Outils > Obtenir les outils et fonctionnalités...** , qui ouvre Visual Studio Installer. Choisissez la charge de travail **Développement .NET Desktop**, puis choisissez **Modifier**.

Visual Studio crée le projet.

### 2. Dans *Program.cs*, remplacez le code suivant


```
C#
```

 Copier

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace get_started_debugging
{
    class Program
    {
        static void Main(string[] args)
        {
        }
    }
}
```

par le code suivant :

C#	 Copier
<pre>using System; using System.Collections.Generic;  public class Shape {     // A few example members     public int X { get; private set; }     public int Y { get; private set; }     public int Height { get; set; }     public int Width { get; set; }      // Virtual method     public virtual void Draw()     {         Console.WriteLine("Performing base class drawing tasks");     } }  class Circle : Shape {     public override void Draw()     {         // Code to draw a circle...         Console.WriteLine("Drawing a circle");         base.Draw();     } }  class Rectangle : Shape {     public override void Draw()     {</pre>	

```
// Code to draw a rectangle...
Console.WriteLine("Drawing a rectangle");
base.Draw();
}
}

class Triangle : Shape
{
    public override void Draw()
    {
        // Code to draw a triangle...
        Console.WriteLine("Drawing a triangle");
        base.Draw();
    }
}


class Program
{
    static void Main(string[] args)
    {
        var shapes = new List<Shape>
        {
            new Rectangle(),
            new Triangle(),
            new Circle()
        };

        foreach (var shape in shapes)
        {
            shape.Draw();
        }


        // Keep the console open in debug mode.
        Console.WriteLine("Press any key to exit.");
        Console.ReadKey();
    }
}

/* Output:
    Drawing a rectangle
    Performing base class drawing tasks
    Drawing a triangle
    Performing base class drawing tasks
    Drawing a circle
    Performing base class drawing tasks
*/
```


## Démarrez le débogueur !

1. Appuyez sur **F5 (Déboguer > Démarrer le débogage)** ou sur le bouton **Démarrer le débogage**  dans la barre d'outils Débogage.

**F5** démarre l'application avec le débogueur attaché au processus de l'application, mais jusqu'à présent, nous n'avons rien fait de spécial pour examiner le code. L'application se charge juste et vous voyez la sortie de la console.

cmd	 Copier
<pre>Drawing a rectangle Performing base class drawing tasks Drawing a triangle Performing base class drawing tasks Drawing a circle Performing base class drawing tasks</pre>	

Dans ce tutoriel, nous examinons cette application plus en détail avec le débogueur et nous regardons les fonctionnalités du débogueur.

2. Arrêtez le débogueur en appuyant sur le bouton d'arrêt rouge .


## Définir un point d'arrêt et démarrer le débogueur

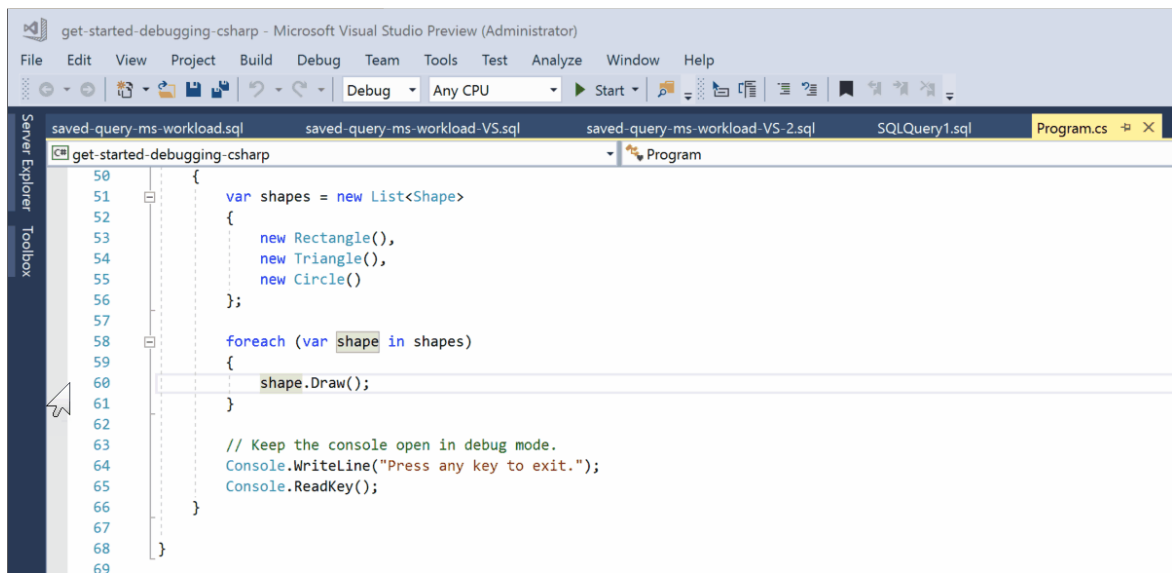
1. Dans la boucle `foreach` de la fonction `Main`, définissez un point d'arrêt en cliquant dans la marge gauche de la ligne de code suivante :

```
shape.Draw()
```

Un cercle rouge apparaît là où vous avez défini le point d'arrêt.

Les points d'arrêt constituent une fonctionnalité élémentaire et essentielle de toute procédure de débogage fiable. Quand vous définissez un point d'arrêt, Visual Studio interrompt l'exécution du code à l'emplacement du point d'arrêt pour vous permettre d'examiner les valeurs des variables, le comportement de la mémoire ou encore la bonne exécution ou non d'une branche de code.

2. Appuyez sur **F5** ou cliquez sur le bouton **Démarrer le débogage** , l'application démarre et le débogueur s'exécute jusqu'à la ligne de code où vous avez défini le point d'arrêt.



La flèche jaune représente l'instruction sur laquelle le débogueur s'est mis en pause, ce qui interromp également l'exécution de l'application au même point (cette instruction n'a pas encore été exécutée).

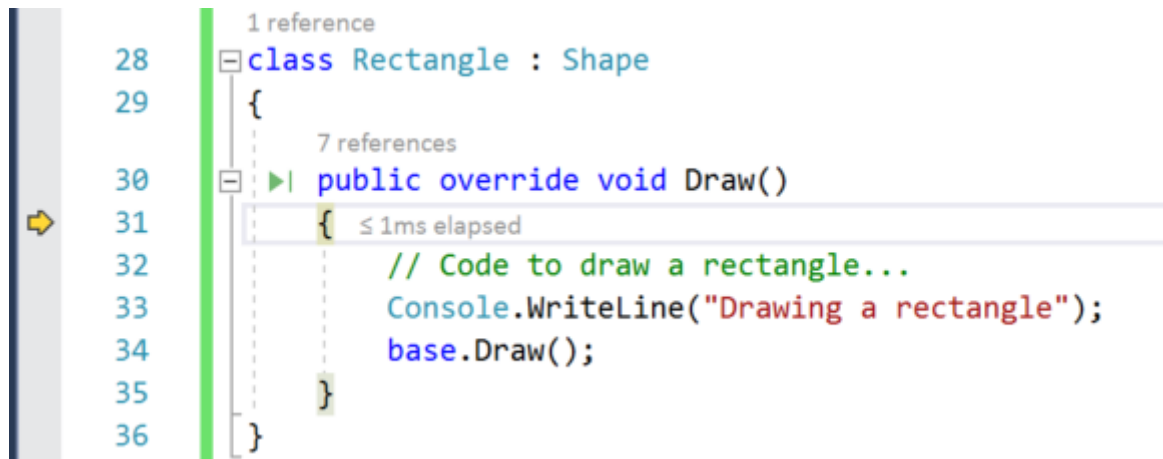
Si l'application ne s'exécute pas encore, **F5** démarre le débogueur et s'arrête au premier point d'arrêt. Sinon, **F5** continue l'exécution de l'application jusqu'au point d'arrêt suivant.

Les points d'arrêt sont une fonctionnalité pratique quand vous savez quelle ligne de code ou section de code vous voulez examiner en détail.

## Parcourir le code dans le débogueur avec les commandes d'exécution pas à pas

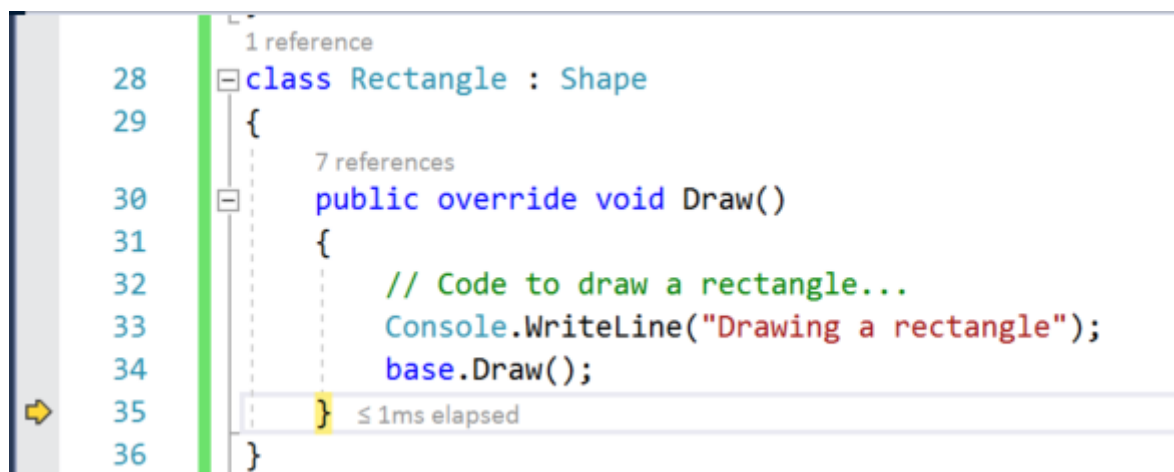
Nous utilisons ici principalement des raccourcis clavier, car c'est un bon moyen d'exécuter rapidement votre application dans le débogueur (les commandes équivalentes, comme les commandes des menus, sont indiquées entre parenthèses).

1. Alors que l'exécution est mise en pause dans l'appel de la méthode `shape.Draw` dans la méthode `Main`, appuyez sur **F11** (ou choisissez **Déboguer > Pas à pas détaillé**) pour avancer dans le code de la classe `Rectangle`.




F11 est la commande **Pas à pas détaillé** : elle fait avancer l'exécution de l'application une instruction à la fois. F11 est un bon moyen pour examiner le flux de l'exécution de la façon la plus détaillée. (Pour avancer plus rapidement dans le code, il existe d'autres options, que nous allons vous montrer.) Par défaut, le débogueur ignore le code non-utilisateur (si vous voulez plus d'informations, consultez [Uniquement mon code](#)).

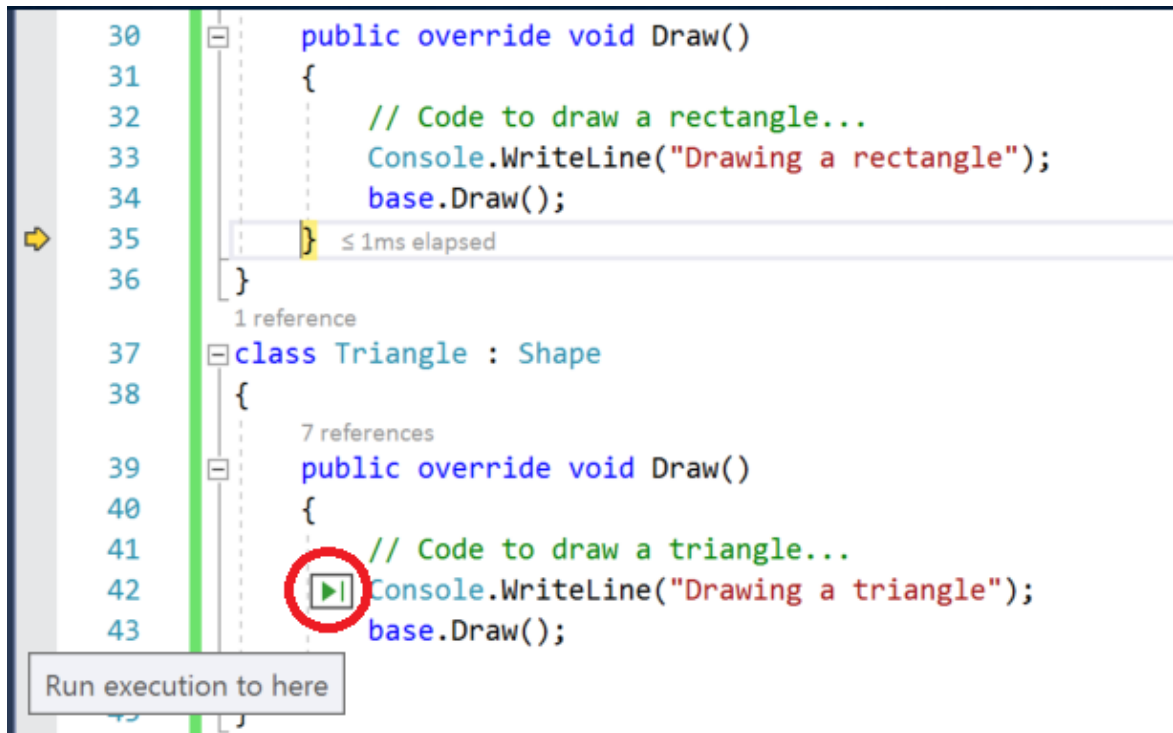
2. Appuyez plusieurs fois sur **F10** (ou choisissez **Débuguer > Pas à pas principal**) jusqu'à ce que le débogueur s'arrête à l'appel de la méthode `base.Draw`, puis appuyez sur **F10** encore une fois.



Notez que cette fois, le débogueur n'effectue pas de pas à pas détaillé dans la méthode `Draw` de la classe de base (`Shape`). **F10** fait avancer le débogueur sans effectuer de pas à pas détaillé dans les fonctions ou les méthodes du code de votre application (le code s'exécute néanmoins). En appuyant sur **F10** sur l'appel de méthode `base.Draw` (au lieu de **F11**), nous avons ignoré le code d'implémentation de `base.Draw` (qui potentiellement ne nous intéresse pas pour l'instant).

## Parcourir le code avec Exécuter jusqu'au clic

1. Dans l'éditeur de code, faites défiler vers le bas et placez le curseur sur la méthode `Console.WriteLine` dans la classe `Triangle` jusqu'à ce que le bouton vert **Exécuter jusqu'au clic**  apparaisse à gauche. L'info-bulle du bouton indique « Lancer l'exécution jusqu'ici ».



#### Notes

Le bouton **Exécuter jusqu'au clic** est une nouveauté de Visual Studio 2017. Si vous ne voyez pas le bouton avec la flèche verte, utilisez à la place **F11** dans cet exemple pour faire avancer le débogueur jusqu'au bon endroit.

2. Cliquez sur le bouton **Exécuter jusqu'au clic** .

L'utilisation de ce bouton revient à définir un point d'arrêt temporaire. **Exécuter jusqu'au clic** est pratique pour examiner rapidement une zone visible du code d'application (vous pouvez cliquer dans n'importe quel fichier ouvert).

Le débogueur avance jusqu'à l'implémentation de la méthode `Console.WriteLine` pour la classe `Triangle`.

Alors que l'application est mise en pause, vous remarquez une faute de frappe ! La sortie « Drawing a trangle » est mal orthographiée. Nous pouvons la corriger directement ici pendant l'exécution de l'application dans le débogueur.

## Modifier le code et continuer le débogage



1. Cliquez dans « Drawing a triangle » et corrigez en remplaçant « trangle » par « triangle ».
2. Appuyez une fois sur **F11** : vous voyez que le débogueur avance à nouveau.

#### ⓘ Notes

Selon le type de code que vous modifiez dans le débogueur, vous pouvez voir un message d'avertissement. Dans certains scénarios, vous devez recompiler le code avant de pouvoir continuer.

## Pas à pas sortant


Supposons que vous avez terminé d'examiner la méthode `Draw` de la classe `Triangle` et que vous voulez quitter la fonction, mais rester dans le débogueur. Vous pouvez faire cela avec la commande **Pas à pas sortant**.

1. Appuyez sur **Maj + F11** (ou **Déboguer > Pas à pas sortant**).

Cette commande reprend l'exécution de l'application (et fait avancer le débogueur) jusqu'au retour de la fonction active.

Vous devez normalement être revenu dans la boucle `foreach` de la méthode `Main`.

## Redémarrer rapidement votre application

Cliquez sur le bouton **Redémarrer**  dans la barre d'outils Débogage (**Ctrl + Maj + F5**).

Quand vous appuyez sur **Redémarrer**, vous gagnez du temps par rapport à l'action consistant à arrêter l'application, puis à redémarrer le débogueur. Le débogueur se met en pause sur le premier point d'arrêt qui est atteint par l'exécution du code.

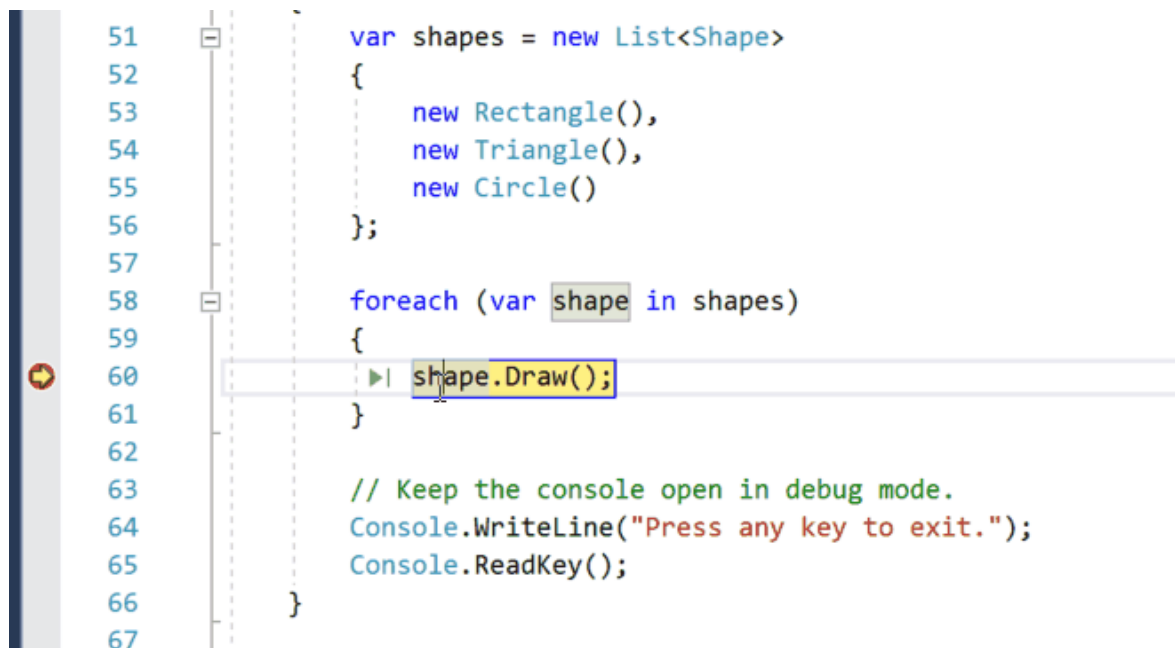
Le débogueur s'arrête à nouveau au niveau du point d'arrêt que vous définissez, à la méthode `shape.Draw()`.

## Inspecter des variables avec des bulles d'informations (datatips)

Les fonctionnalités qui vous permettent d'inspecter des variables sont parmi les plus pratiques du débogueur : vous pouvez faire cela de différentes façons. Souvent, quand

vous essayez de déboguer un problème, vous essayez de déterminer si les variables stockent les valeurs que vous prévoyez à un moment donné.

1. Alors que l'exécution est mise en pause sur la méthode `shape.Draw()`, placez le curseur sur l'objet `shape`. Vous voyez alors sa valeur de propriété par défaut, le type d'objet `Rectangle`.
2. Développez l'objet `shape` pour voir toutes ses propriétés (notamment la propriété `Height`, qui a la valeur 0).
3. Appuyez plusieurs fois sur **F10** (ou choisissez **Débuguer** > **Pas à pas principal**) pour itérer une fois la boucle `foreach`, en effectuant à nouveau une pause sur `shape.Draw()`.
4. Placez une nouvelle fois le curseur sur l'objet `shape`. Cette fois-ci, vous voyez un nouvel objet de type `Triangle`.



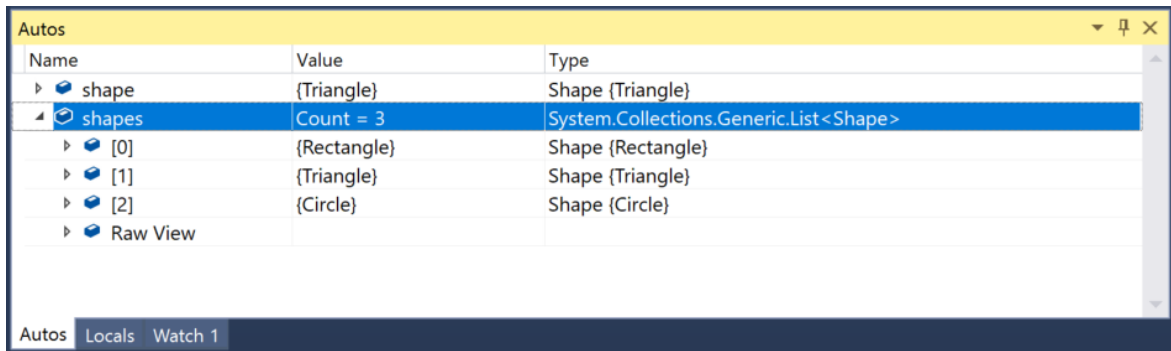
Souvent, lors du débogage, vous voulez un moyen rapide de vérifier les valeurs des propriétés sur des variables pour voir si elles stockent bien les valeurs prévues. Les bulles d'informations (« data tips ») sont un bon moyen de le faire.

## Inspecter des variables avec les Fenêtres Automatique et Variables locales

1. Examinez la fenêtre **Automatique** en bas de l'éditeur de code.

Si elle est fermée, ouvrez-la pendant que l'exécution est mise en pause dans le débogueur en choisissant **Débuguer** > **Windows** > **Automatique**.

## 2. Développez l'objet `shapes`.



Dans la fenêtre **Automatique**, vous voyez des variables et leur valeur actuelle. La fenêtre **Automatique** montre toutes les variables utilisées dans la ligne active ou la ligne précédente (consultez la documentation pour les comportements selon le langage).

3. Ensuite, examinons la fenêtre **Variables locales**, sous un onglet à côté de la fenêtre **Automatique**.

La fenêtre **Variables locales** montre les variables qui se trouvent dans l'[étendue](#) actuelle, c'est-à-dire le contexte d'exécution actif.

## Définir un espion

1. Dans la fenêtre principale de l'éditeur de code, cliquez sur l'objet `shapes` et choisissez **Ajouter un espion**.

La fenêtre **Espion** s'ouvre en bas de l'éditeur de code. Vous pouvez utiliser une fenêtre **Espion** pour spécifier une variable (ou une expression) que vous voulez observer.

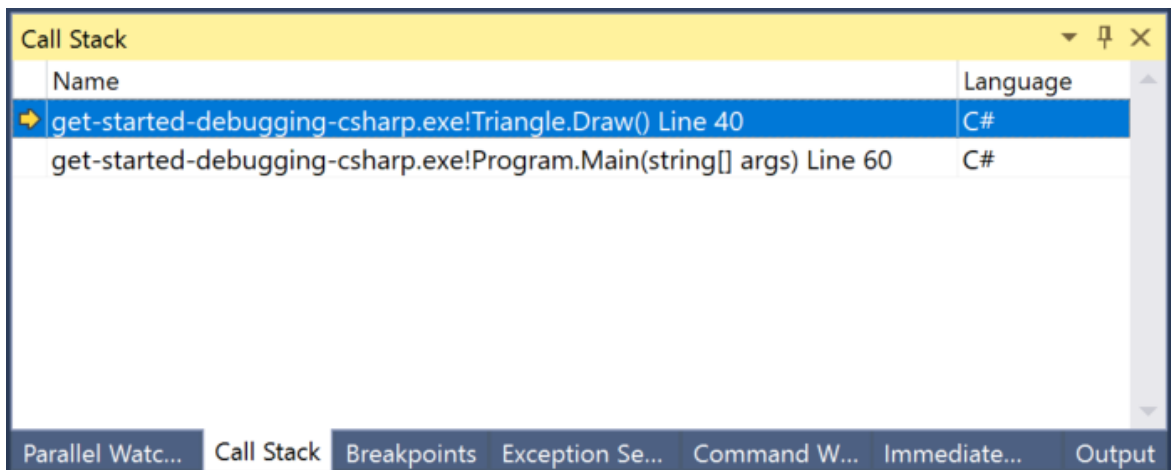
Vous avez maintenant un espion défini sur l'objet `shapes` et vous pouvez voir sa valeur changer au fil de votre déplacement dans le débogueur. Contrairement à d'autres fenêtres de variables, la fenêtre **Espion** montre toujours les variables que vous observez (elles apparaissent en grisé quand elles sont en dehors de l'étendue).

## Examiner la pile des appels

1. Alors que l'exécution est mise en pause dans la boucle `foreach`, cliquez sur la fenêtre **Pile des appels** qui est ouverte par défaut dans le volet inférieur droit.

Si elle est fermée, ouvrez-la pendant que l'exécution est mise en pause dans le débogueur en choisissant **Déboguer** > **Windows** > **Pile des appels**.

- Appuyez plusieurs fois sur **F11** jusqu'à ce que le débogueur fasse une pause dans la méthode `Base.Draw` de la classe `Triangle` dans l'éditeur de code. Regardez la fenêtre **Pile des appels**.



La fenêtre **Pile des appels** montre l'ordre dans lequel les méthodes et les fonctions sont appelées. La ligne du haut montre la fonction active (méthode `Triangle.Draw` dans cette application). La deuxième ligne montre que `Triangle.Draw` a été appelée à partir de la méthode `Main`, etc.

#### ⓘ Notes

La fenêtre **Pile des appels** est similaire à la perspective Débogage dans certains IDE, comme Eclipse.

La pile des appels est un bon moyen d'examiner et de comprendre le flux d'exécution d'une application.

Vous pouvez double-cliquer sur une ligne de code pour accéder à ce code source ; ceci change également l'étendue active inspectée par le débogueur. Cette action ne fait pas avancer le débogueur.

Vous pouvez également utiliser les menus contextuels de la fenêtre **Pile des appels** pour faire d'autres choses. Par exemple, vous pouvez insérer des points d'arrêt dans des fonctions spécifiées, faire avancer le débogueur avec **Exécuter jusqu'au curseur** et aller examiner le code source. Pour plus d'informations, voir [Procédure : examiner la pile des appels](#).

## Changer le flux d'exécution

- Alors que le débogueur est mis en pause dans l'appel de la méthode `Circle.Draw`, utilisez la souris pour sélectionner la flèche jaune (pointeur d'exécution) sur la

gauche, puis déplacez la flèche jaune d'une ligne vers le haut, où est fait l'appel de la méthode `Console.WriteLine`.

## 2. Appuyez sur **F11**.

Le débogueur réexécute la méthode `Console.WriteLine` (vous voyez ceci dans la sortie de la fenêtre de console).

En changeant le flux d'exécution, vous pouvez effectuer des opérations comme tester d'autres chemins d'exécution du code ou réexécuter du code sans devoir redémarrer le débogueur.

### **Avertissement**

Vous devez rester prudent avec cette fonctionnalité, vous pouvez voir un avertissement dans l'info-bulle. Vous pouvez aussi en voir d'autres. Le fait de déplacer le pointeur ne peut pas rétablir votre application à un état antérieur.

## 3. Appuyez sur **F5** pour poursuivre l'exécution de l'application.

Félicitations ! Vous avez terminé ce didacticiel.

# Étapes suivantes

Dans ce tutoriel, vous avez découvert comment démarrer le débogueur, parcourir le code pas à pas et inspecter des variables. Vous pouvez obtenir une présentation générale des fonctionnalités du débogueur et suivre des liens qui donnent accès à plus d'informations.

[Présentation du débogueur](#)

---

### Cette page est-elle utile ?

 Oui  Non

---