ويژگي هاي سي شارپ ۳,۰

مقدمه

زبان برنامه نويسي سي شارپ با نسخه ۱,۰ به جهان برنامه نويسي عرضه شد. Visual Studio .NET مجهز به كامپايلر آن بود و برنامه نويسان مي توانستند از ويژگي هاي نسخه اوليه استفاده كنند. نسخه دوم سي شارپ همراه با ويژوال استوديوي ۲۰۰۵ ارائه شد و توانايي را به نسخه قبلي افزود. مدتي است كه شركت مايكروسافت به صورت آزمايشي آخرين نسخه زبان سي شارپ يعني نسخه ۳,۰ آن را در اختيار برنامه نويسان سراسر جهان قرار داد. اين نسخه قرار است به صورت نهايي در سال ۲۰۰۸ به همراه Visual Studio 2008 ارائه شود. در اين مقاله سعي خواهيم كرد مهم ترين ويژگي هاي افزوده شده به نسخه سي شارپ ۳,۰ را مورد بررسي قرار دهيم.

C# 3.0

شايد اين سوال در ذهن بسياري از كساني كه در حال فراگيري سي شارپ هستند، مطرح باشد كه چرا سي شارپ با اين سرعت در حال دگرگوني است؟ دو سال از ارائه شدن نخسه ۲,۰ نمي گذرد كه نسخه ۳,۰ هم ارائه شده است! در پاسخ بايد گفت كه اين روند كاملا طبيعي بوده و هيچ گونه مشكلي براي برنامه نويساني كه از زبان سي شارپ استفاده مي كنند، ايجاد نمي كند.پايه زبان سي شارپ، نسخه ۱,۰ آن مي باشد. اين نسخه تمامي كمات كليدي و ساختارهاي اصلي زبان را مشخص مي كند. آنچه كه در نسخه هاي بعدي به زبان اضافه مي شود تنها ويژگي هاي جديد است كه موجب تسهيل امر كدنويسي مي شود.

قبل از ارائه شدن نسخه جديد يك زبان، ويژگي ها و خصوصيات آن به صورت فايل متني (file Specification) در اختيار جامعه برنامه نويسي قرار مي گيرد. ويژگي ها و خصوصيات، نحوه عملكرد نسخه جديد را بيان مي كنند.كساني كه قصد داشته باشند با ويژگي هاي نسخه ۳,۰ سي شارپ آشنا شوند مي توانند فايل Specification ۳,۰ #C را مطالعه كنند. ويژگي هاي جديد سي شارپ ۳,۰ كه در اين فايل بيان شده اند، عبارت اند از:

- Implicitly typed local variables, which permit the type of local variables to be inferred from the expressions used to initialize them.
- Extension methods, which make it possible to extend existing types and constructed types with additional methods.
- Lambda expressions, an evolution of anonymous methods that provides improved type inference and conversions to both delegate types and expression trees.
- Object initializers, which ease construction and initialization of objects.
- Anonymous types, which are tuple types automatically inferred and created from object initializers.
- Implicitly typed arrays, a form of array creation and initialization that infers the element type of the array from an array initializer.
- Query expressions, which provide a language integrated syntax for queries that is similar to relational and hierarchical query languages such as SQL and XQuery.
- Expression trees, which permit lambda expressions to be represented as data (expression trees) instead of as code (delegates).

در ادامه اين مقاله، مهم ترين ويژگي هاي ذكر شده را شرح خواهيم داد.

Implicitly typed local variables – متغيرهاي ضمني محلي

سي شارپ ۳,۰ كلمه كليدي جديد var را معرفي مي كند كه به كمك آن برنامه نويسان قادر خواهند بود متغيرهاي محلي خود را بدون ذكر صريح نوع آن ها، تعريف كنند.با توجه به اين ويژگي جديد، تعريف يك رشته و يا مقدار عددي به صورت زير امكانپذير است:

```
namespace CS3_Test
{
    class Program
    {
        static void Main(string[] args)
        {
            var string_value = "Hello C# 3.0!";
            var int_value = 3;
        }
    }
}
```

يكي از ويژگي هاي اصلي زبان سي شارپ، Strong Type بودن آن است. Strong Type بودن زبان به اين معناست كه با اعلان يك متغير، نوع آن صريحا بايد توسط برنامه نويس مشخص شود. آيا اضافه شدن ويژگي جديد، منافاتي با Strong Type بودن سي شارپ دارد؟در پاسخ بايد گفت كه تعريف متغيرهاي محلي به صورت ضمني با استفاده از كلمه كليدي var هيچ گونه منافاتي با Strong Type بودن سي شارپ ندارد. چون برنامه نويس مي بايست نوع متغير را به هنگام اعلان آن صريحا مشخص كند.نوع متغير پس از اولين اعلان تا اتمام حوزه تعريف آن تغيير نخواهد كرد و هر گونه تلاش براي تغيير نوع با خطا مواجه خواهد شد.با توجه به موارد بيان شده دو اعلان زير نامعتبر هستند:

```
    class Program
    {
        static void Main(string[] args)
        {
Error no. 1 → var string_value;
Error no. 2 → var int_value = null;
        }
    }
}
```

Error List

2 Errors | 0 Warnings | 0 Messages

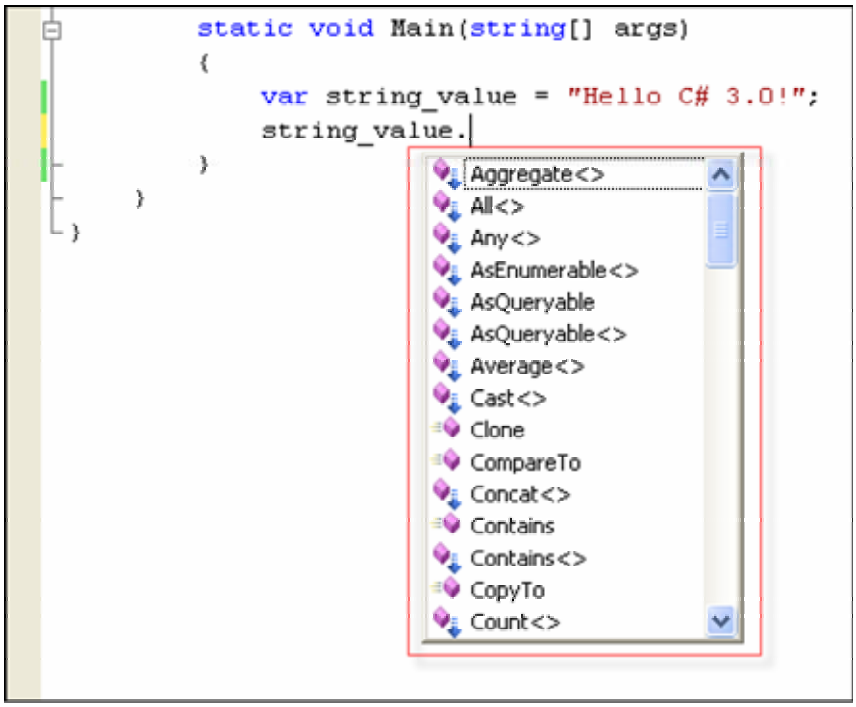
	Description
1	Implicitly typed locals must be initialized
2	Cannot assign '<null>' to an implicitly typed local

دقت كنيد كه استفاده از var تنها در تعريف متغيرهاي محلي امكانپذير است. در اعلان متغيرها به صورت سراسري، پارامترهاي توابع و مقادير بازگشتي نمي توان از var استفاده كرد.

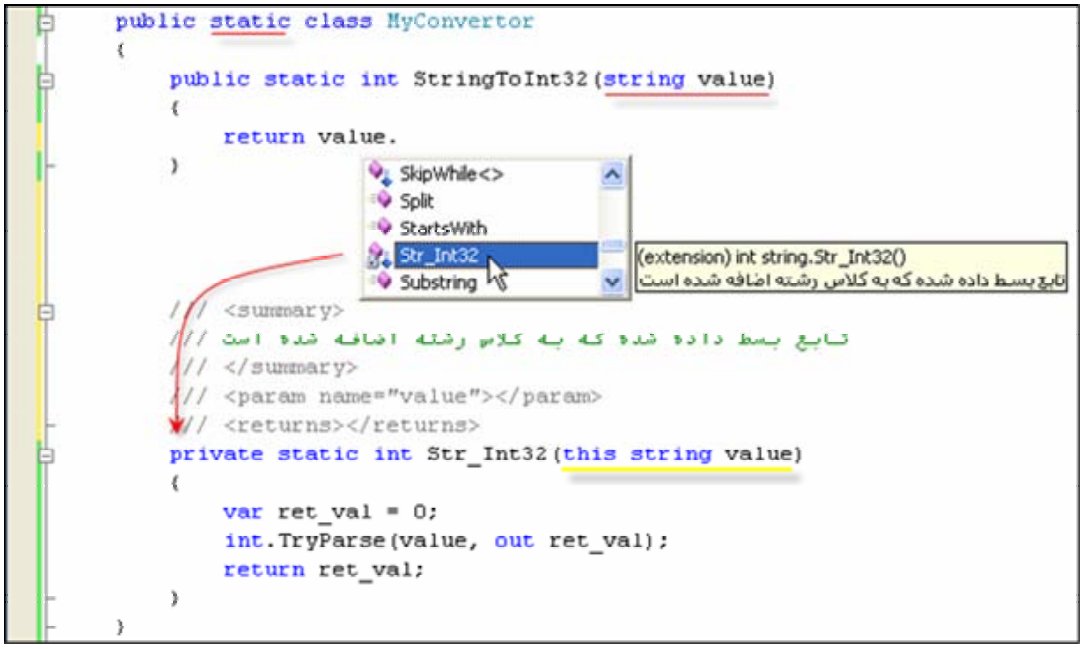
چرا var ؟ اين ويژگي آزادي عملي بيشتري براي كار با متغيرهاي محلي در اختيار برنامه نويس قرار مي دهد. سناريويي را در نظر بيگيريد كه يك تابع تحت شرايطي، مقادير از انواع مختلف را برگرداند. در اين صورت بدون درگير شدن با casting و تبديل نوع مي توان با تعريف متغير ضمني محلي هر نوعي را كه تابع برمي گرداند، در اختيار داشت.

Extension methods – توابع بسط داده شده

زبان سي شارپ كلمه كليدي sealed را براي اين منظور ارائه كرد كه امكان ارث بري از يك كلاس را صلب كند. يعني با اضافه شدن اين كلمه كليدي به ابتداي تعريف كلاس، امكان ارث بري از آن غير ممكن مي شود. سي شارپ ۳,۰ ويژگي جديدي را در اختيار برنامه نويسان قرار مي دهد به اين صورت كه مي توان هر نوع كلاسي – حتي كلاس هاي مهر شده با Sealed را با استفاده از Extension methods بسط داد. به مثال زير توجه كنيد:



اعضاي كلاس String در آن نشان داده شده اند! كلاس String از جمله كلاس هايي است كه با ارث بري قابل بسط دادن نمي باشد. اما اينك توابع زيادي به عنوان Extension methods به آن اضافه شده اند كه با آيكون  مشخص مي شوند. تنها نكته مهم اين است كه Extension methods فقط به كلاس هاي Static اضافه مي شوند. مانند مثال زير :



در اين مثال كلاس استاتيك MyConvertor به همراه تابع عضو آن StringToInt32 تعريف شده است كه مقادير رشته اي را به مقادير عددي تبديل مي كند.دقت كنيد كه MyConvertor، يك تابع بسط داده شده به كلاس String اضافه مي كند.نحو (syntax) تعريف تابع بسط داده شده با زيرخط زرد رنگ مشخص شده است.در تعريف تابع بسط داده شده بايد از كلمه كليدي this به صورت قراردادي استفاده كرد.پس از كلمه كليدي بايد نوعي را مشخص كرد كه دسترسي به تابع از طريق آن امكانپذير خواهد بود.(در مثال بالا تابع Str_Int32 از طريق كلاس string قابل دسترسي خواهد بود) پس از تعريف تابع بسط داده شده، مي توان در مجموعه توابع كلاس String از آن استفاده كرد كه اين مورد در شكل به خوبي نشان داده شده است.

چرا methods Extension؟ همانطور كه بيان شد بسياري از كلاس هاي ارائه شده در دات نت فريم ورك با برچسب sealed، قابل بسط داده شدن نيستند. كلاس مفيدي مثل List را در نظر بگيريد كه از جمله اين كلاس هاست. با استفاده از ويژگي معرفي شده برنامه نويسان قادر خواهند بود با اضافه كردن توابع دلخواه خود به اين كلاس ها، آنها را بنا به نياز خود بسط دهند.

عبارات لامبدا – expressions Lambda

يكي از ويژگي هايي كه سي شارپ ۳,۰ ارائه كرد، توانايي تعريف توابع به صورت Inline بود كه اين ويژگي با عنوان توابع بي نام (anonymous methods) شناخته مي شود. توابع بي نام در پاره اي مواقع بسيار مفيدند. اما نحو (syntax) به كارگيري آنها دشوار مي باشد. عبارات لامبدا ويژگي توابع بي نام را دارند اما با نحو ساده تري در سي شارپ ۳,۰ معرفي شده اند. به نمونه اي از عبارات لامبداي تعريف شده توجه كنيد:

!Error

```
static void Main(string[] args)
{
    (int x) => x + 1;          // explicitly typed parameter
    (y, z) => y * z;           // implicitly typed parameter
}
```

تعريف عبارات لامبدا از نحو (syntax) خاصي پيرو مي كند. همانطور كه در شكل مشاهده مي كنيد، پارامترهاي تابع هم به صورت صريح و هم به صورت ضمني قابل بيان اند. كلمه return به صورت ضمني حذف شده است. تابع معادل عبارت لامبداي اول به شكل زير مي باشد:

```
int Fn(int x)
{
    return x + 1;
}
```

ليست پارامترها و بدنه عبارت لامبدا توسط => از هم جدا مي شوند. در صورتي كه تعريف عبارت لامبدا بيشتر از يك خط كد باشد مي توان بدنه آن را با استفاده از {} نشان داد مانند شكل زير:

```
static void Main(string[] args)
{
    (int x) => { x + 1; return x * x; };
}
```

در ادامه يك مثال عملي از عبارات لامبدا بيان مي شود. كلاس MyMath را در نظر بگيريد. يك نماينده و يك تابع در داخل اين كلاس تعريف شده اند:

```
public class MyMath
{
    delegate int MathFn(int a, int b);

    public MyMath()
    {

    }

    public int MathInvoker(MathFn Operation,int a,int b)
    {
        return Operation(a, b);
    }
```

تابع MathInvoker براي اين منظور در نظر گرفته شده است كه از طريق نماينده Operation كه از نوع MathFn مي باشد، يكي از چهار عملي اصلي را دريافت كرده، و با استفاده از آرگومان هاي ارسالي، تابع مربوطه را فراخواني كند.به فراخواني هاي زير دقت كنيد:

!Error

```
static void Main(string[] args)
{
    MyMath myMath = new MyMath();

    var a = 5;
    var b = 5;

    //Anonymous Method
    var res1 = myMath.MathInvoker(delegate(int val1, int val2)
    {
        return val1 * val2;
    }, a, b);

    //Lambda Expression
    var res2 = myMath.MathInvoker((val1, val2) => val1 * val2, a, b);

    Console.WriteLine("By Anonymous Method :");
    Console.WriteLine(res1);

    Console.WriteLine("By Lambda Expression :");
    Console.WriteLine(res2);
}
```

تابع MathInvoker اول با پارامتر تابع بي نام و سپس باعبارت لامبدا فراخواني شده است.سادگي كار با عبارت لامبدا نسبت به تابع بي نام كاملا مشهود مي باشد. (عبارت لامبدا با زير خط آبي مشخص شده است)

چرا expressions Lambda؟ عبارات لامبدا براي جايگزين كردن توابع بي نام ارائه شده اند.نحو ساده تري دارند.در ضمن در Expression trees از آنها استفاده مي شود.

initializers Object – سازنده هاي پيشرفته

ماهيت تمامي برنامه هاي امروزي به گونه اي ست كه با حجم عظيمي از داده ها سرو كار دارند.براي مديريت داده ها، نياز به كلاس هايي ست كه در مهندسي نرم افزار آنها را Entity Types مي ناميم.اين كلاس ها به عنوان بسته هايي از داده ها محسوب مي شوند.معضل فعلي موجود در رابطه با Entity Typeها تعدد سازنده هاي آن ها مي باشد و ممكن است شما نيز با اين مشكل برخورد كرده باشيد.به اين صورت كه در سناريوهاي مختلف، برنامه نويسان مجبور هستند سازنده يك كلاس را به چند شكل سربارگذاري كنند.سي شارپ ۳,۰ راه چاره اي فوق العاده براي اين مشكل ارائه مي دهد.initializer Object حالت پيشرفته اي از سازنده مي باشد.يك كلاس Entity به نام Person در نظر بگيريد كه داده هاي زير را بسته بندي (پكيج) مي كند:

```
public class Person
{
    private string _firstName;

    public string FirstName
    {
        get { return _firstName; }
        set { _firstName = value; }
    }

    private string _lastName;

    public string LastName
    {
        get { return _lastName; }
        set { _lastName = value; }
    }

    private int _age;

    public int Age
    {
        get { return _age; }
        set { _age = value; }
    }
}
```

اين كلاس شامل سه متغير بوده و براي هرمتغير شناسه اي تعريف شده است.اينك اين سوالات مطرح مي شوند : سازنده اين كلاس را به چند شكل بايد سربارگذاري كرد؟ سازنده اي كه هر سه متغير را مقداردهي كند؟ شايد در مواردي هر سه متغير در دست نباشد در اين صورت چه سازنده اي بايد فراخواني شود؟ سي شارپ ۳,۰ راه حل زير را ارائه مي دهد.فرض كنيد بخوانيم نمونه اي ازكلاس Person را ايجاد كنيم:

```
static void Main(string[] args)
{
    Person person = new Person {
}
}
```


همانطور كه مشاهده مي كنيد،در سي شارپ ۳,۰ به هنگام نمونه سازي، اين امكان در اختيار برنامه نويس قرار مي گيرد كه هر يك از خصيصه هاي موجود در كلاس را به دلخواه و بنا به نياز مقدار دهي كند (فوق العاده است نه؟!) به شكل زير :

```
Person person = new Person
                {
                    Age = 23,
                    FirstName = "Abolfazl",
                    LastName = "Hosnaddinov"
                };
}
```

توجه داشته باشيد كه به جاي () از {} استفاده شده است.در ضمن هيچ سازنده اي در كلاس Person تعريف نشده است!

Anonymous types – تايپ هاي بي نام!

سي شارپ ۳,۰ توابع بي نام را معرفي كرد.سي شارپ ۳,۰ هم انواع بي نام را معرفي مي كند.با استفاده از اين ويژگي برنامه نويسان قادر خواهند بود به صورت Inline انواع دلخواه خود را ايجاد كنند! به نمونه زير توجه كنيد:

!Error

```
static void Main(string[] args)
{
    var anonymousType = new { Name = string.Empty, Age = 0 };
    anonymousType.
}
```


كد ارائه شده، يك نوع بي نام را تعريف مي كند كه از طريق متغير ضمني محلي به نام anonymousType در اختيار قرار مي گيرد.به توضيحات ويژوال استوديو نسبت به نوع ايجاد شده دقت كنيد.همانطور كه مشاهده مي كنيد، تمامي متغيرهايي كه در سازنده پيشرفته ذكر شده اند، در ادامه به عنوان خصيصه هاي نوع بي نام مطرح مي شوند.

چرا types Anonymous؟ انواع بي نام بهترين گزينه براي توليد Entity Typeها مي باشند.همانطور كه گفته شد Entity Typeها فقط حاوي داده ها هستند.بنابراين به بهترين نحو مي توان داده هاي دريافت شده از كاربر را در انواع بي نام بسته بندي كرد.

expressions Query – عبارات كئوري

تيم طراح سي شارپ ويژگي فوق العاده اي را به آن اضافه كرد كه برنامه نويسان را قادر مي سازد نحو (Syntax) زبان هاي پرس و جو مانند SQL و XQuery را با استفاده از سي شارپ پياده سازي كنند! اين ويژگي با نام اختصاري LINQ شناخته مي شود و داراي انواع زير است:

- **LINQ**-to-Objects - talks to in-memory objects
- **LINQ**-to-SQL - talks to SQL Server databases
- **LINQ**-to-XML - talks to hierarchical data represented in XML
- **LINQ**-to-DataSets - talks to DataSet objects and underlying DataTables with their relationships
- **LINQ**-to-Entities - talks to "entities", part of ADO.NET 3.0

نمونه ای ازكاربر LINQ در شكل زير نشان داده شده است :

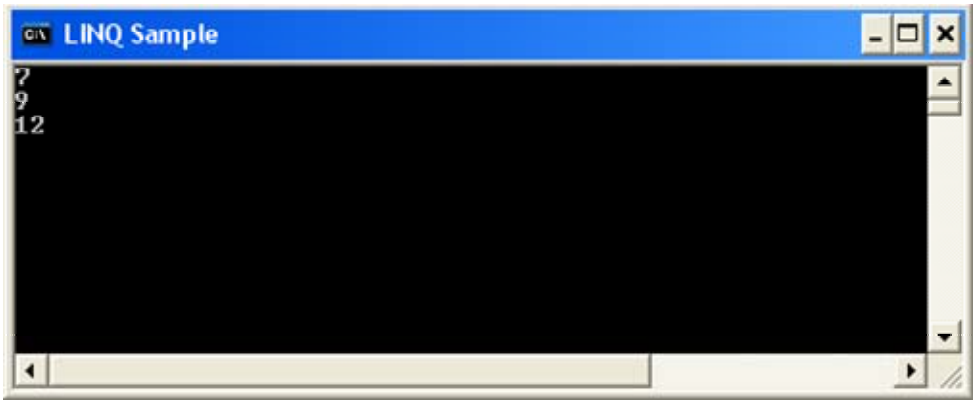
!Error

```
static void Main(string[] args)
{
    var int_array = new int [] { 1, 2, 7, 9, 12 };

    var selective_array = from c in int_array where c > 5 select c;

    foreach (var selected in selective_array)
    {
        Console.WriteLine(selected);
    }
}
```

توضيح : در مثال بالا ابتدا يك آرايه ي intي با مقداردهي اوليه تعريف شده است.سپس با استفاده از دستورات LINQ (كه جز كلمات كليدي سي شارپ محسوب مي شوند)، آرايه اي با اعضاي بزرگتر از ٥ انتخاب و در متغير ضمني محلي selective_array ذخيره مي شود.در نهايت اعضاي selective_array به شكل زير چاپ مي شوند:



در توضيح اين ويژگي جديد به همين يك مثال بسنده مي كنيم چون بيان تمامي جنبه هاي LINQ خود نيازمند نگارش مقاله اي مفصل مي باشد.

دو ويژگي باقيمانده

شش ويژگي سي شارپ ۳,۰ را به طور كامل شرح داديم و اين شش مورد از اهميت بيشتري برخودارند.در مورد Implicitly typed arrays بايد گفت كه اين ويژگي عبارت است از تعريف متغيرهاي آرايه اي با استفاده از كلمه كليدي var .به مثال هاي زير دقت كنيد:

!Error

```
var a = new[] { 1, 10, 100, 1000 };          // int[]
var b = new[] { 1, 1.5, 2, 2.5 };            // double[]
var c = new[] { "hello", null, "world" };    // string[]
var d = new[] { 1, "one", 2, "two" };        // Error
```

trees نيز در رابطه با عبارات لامبدا مطرح مي شود.در توضيحات مربوط به عبارات لامبدا بيان شد كه اين عبارات جايگزيني براي توابع بي نام هستند بنابراين ماهيت delegate ي دارند.اگر بخواهيم به عبارات لامبدا ماهيت داده اي (data) بدهيم،مقوله Expression trees مطرح مي شود.براي كسب اطلاعات كامل تر به مراجع منتشر شده در اين زمينه مراجعه كنيد

جمع بندي

در اين مقاله سي شارپ ۳,۰ را به طور كامل بررسي كرديم. اگر دقت كرده باشيد، بيشتر ويژگي هاي ارائه شده توسط نسخه جديد كار با داده ها را تسهيل بخشيده و با كاهش ميزان كدنويسي، باعث صرفه جويي در وقت خواهند شد.