The fact that a struct can implement an interface is well known and so is the fact that casting a value type into an interface leads to boxing of the value type. This is because methods in interfaces are defined as virtual and to resolve virtual references, vtable (method table) look up is required. Since value types do not have pointers to vtable they are first boxed into a reference type and then the look up happens.

This boxing leads to some performance penalty. See [Rico Mariani's performance quiz](http://blogs.msdn.com/ricom/archive/2005/08/25/456446.aspx)for an example.

The fact that such boxing takes place on casting to interfaces can lead to subtle issues in code. Consider the following

**interface** IPromotion {

void promote();

}

**struct** Employee : IPromotion {

public string Name;

public int JobGrade;

**public void promote() {**

**JobGrade++;**

**}**

public Employee(string name, int jobGrade) {

this.Name = name;

this.JobGrade = jobGrade;

}

public override string ToString() {

return string.Format("{0} ({1})", Name, JobGrade);

}

}

class Program

{

static void Main(string[] args)

{

Employee employee = new Employee("Cool Guy", 65);

**IPromotion p = employee;**

Console.WriteLine(employee);

**p.promote();**

Console.WriteLine(employee);

}

}

Here the output would be   
Cool Guy (65)  
Cool Guy (65)  
So even after calling p.promote() the value of JobGrade in employee does not increase. The reason is that on implicitly casting employee to IPromotion p a temporary boxed object is created andp.promote updates the JobGrade in this temporary object and not on original employee value type. Since after this usage the temporary boxed object is not refered to, the update to it is lost and it is garbage collected.

If however we change the code and make Employee a class the output would become as expected

Cool Guy (65)  
Cool Guy (**66**)

The reason being that now the boxing does not take place and the update happens on the original employeeobject.