Hidden Gems in System.Runtime.CompilerServices

Bart J.F. De Smet
bartde@outlook.com

pluralsight
hardcore dev and IT training

What's in a Name?

- **System.Runtime.CompilerServices**
  - Collection of types used by compilers
- **Examples**
  - Dynamic typing (C# 4.0)
    - CallSite and CallSite<T> used for dynamic dispatch
    - DynamicAttribute
  - Extension methods (C# 3.0)
    - ExtensionAttribute
  - Async methods (C# 5.0)
    - IAsyncStateMachine, INotifyCompletion, ICriticalNotifyCompletion
    - TaskAwaiter<T>, TaskAwaiter, etc.
  - Caller info attributes (C# 5.0)
    - CallerMemberNameAttribute, CallLineNumberAttribute, CallFilePathAttribute
  - Friend assemblies, type forwarders (.NET 2.0)
    - InternalsVisibleToAttribute
    - TypeForwardedToAttribute, TypeForwardedFromAttribute

ConditionalWeakTable<TKey, TValue>

- **Dictionary to bind values to an object**
  - Keys are not kept alive by the dictionary entry
    - Similar to weak references
    - Not the same as an IDictionary<WeakReference<TKey>, TValue>
  - Can be used to build attached properties, fields, etc.
  - Key equality based on object reference
    - Class constraint on TKey and TValue

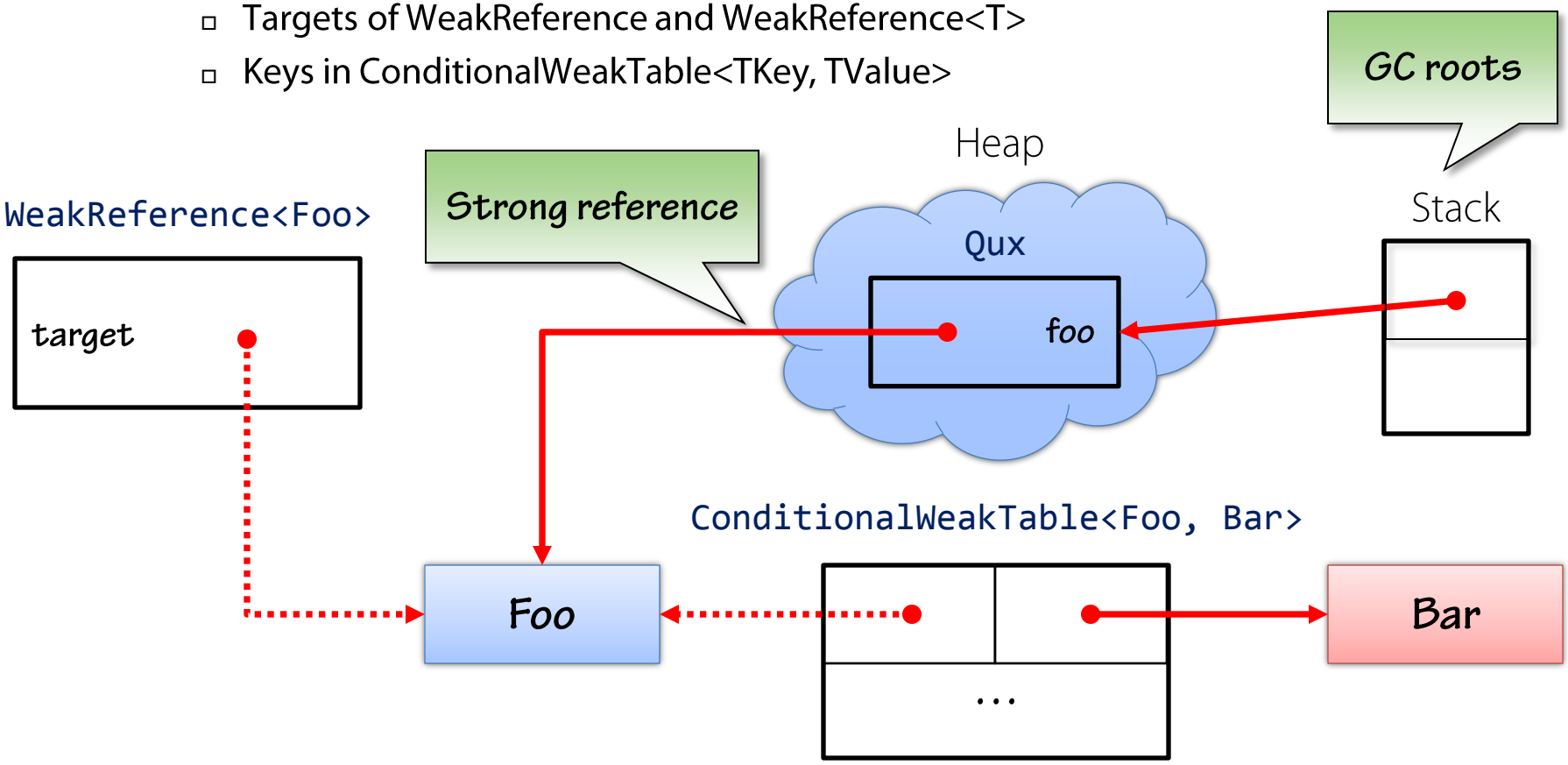
Not a full-fledged dictionary

```
class ConditionalWeakTable<TKey, TValue> where TKey : class
                                         where TValue : class {
  public void Add(TKey key, TValue value);
  public bool Remove(TKey key);
  public bool TryGetValue(TKey key, out TValue value);
  public TValue GetOrCreateValue(TKey key);
  public TValue GetValue(TKey key, CreateValueCallback callback);
}
```

Atomic get or add method

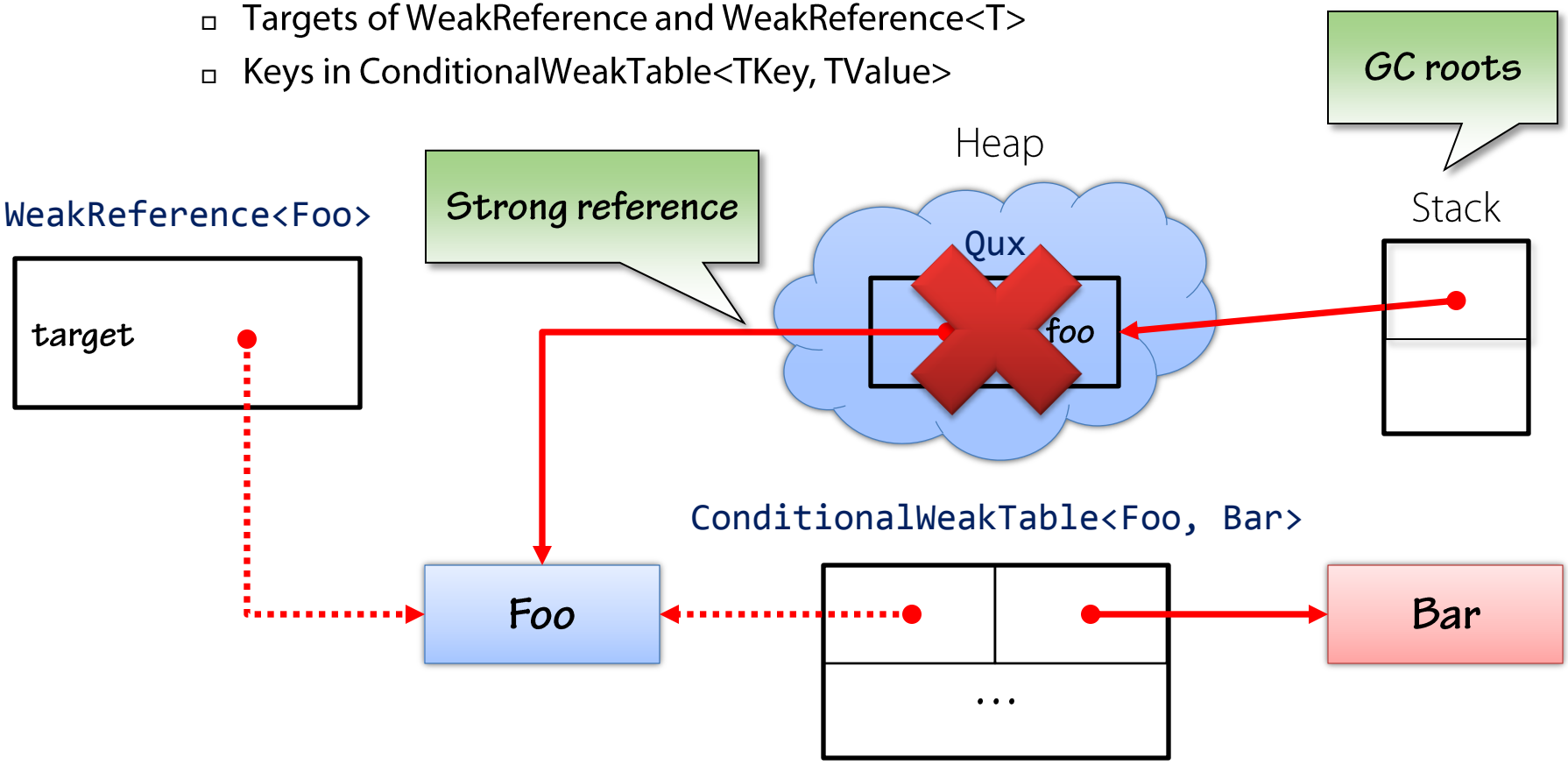
ConditionalWeakTable<TKey, TValue>

- **Object lifetimes**
  - Strong references are the default
  - Weak references
    - Targets of WeakReference and WeakReference<T>
    - Keys in ConditionalWeakTable<TKey, TValue>



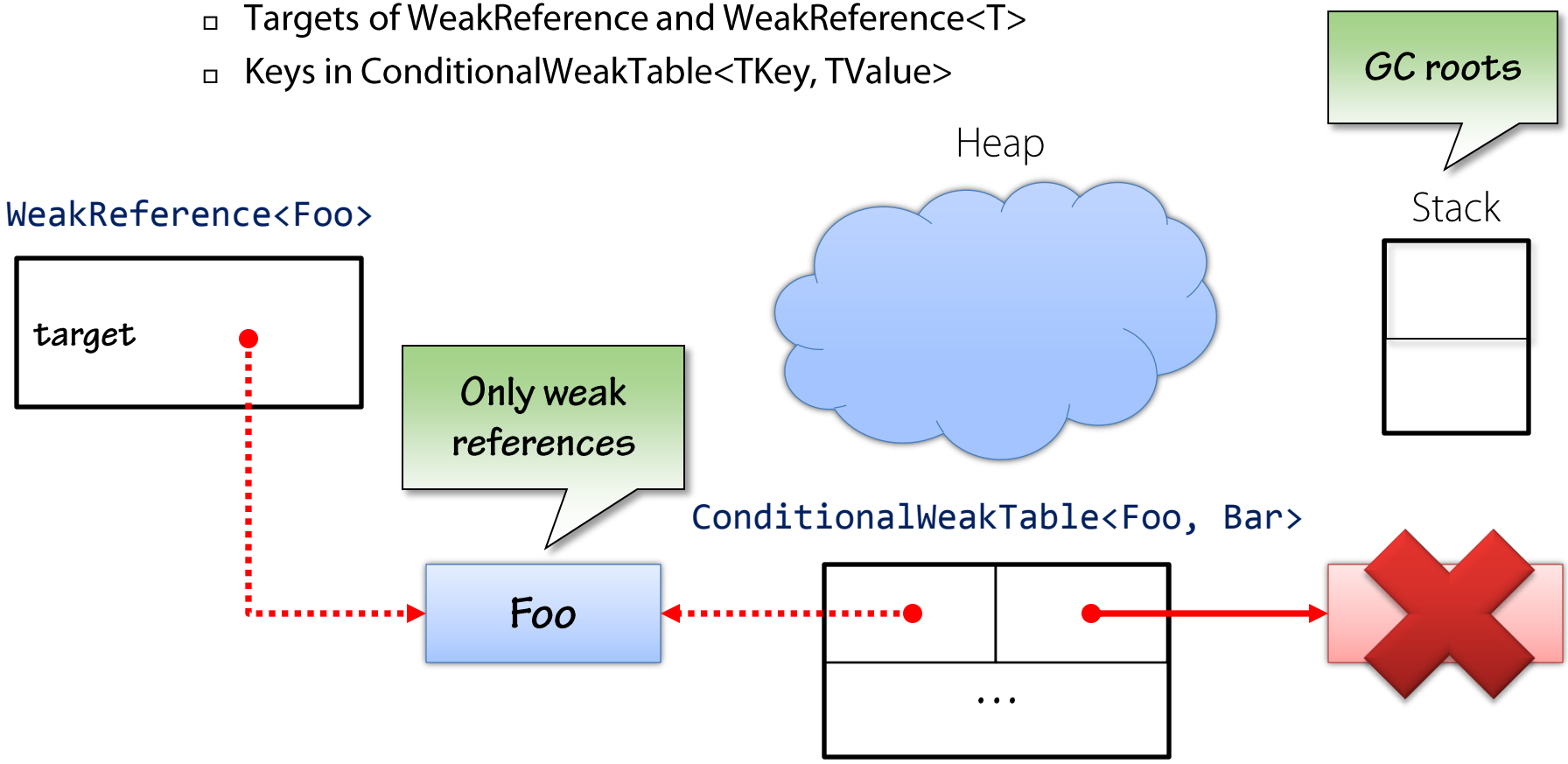
ConditionalWeakTable<TKey, TValue>

- **Object lifetimes**
  - Strong references are the default
  - Weak references
    - Targets of WeakReference and WeakReference<T>
    - Keys in ConditionalWeakTable<TKey, TValue>



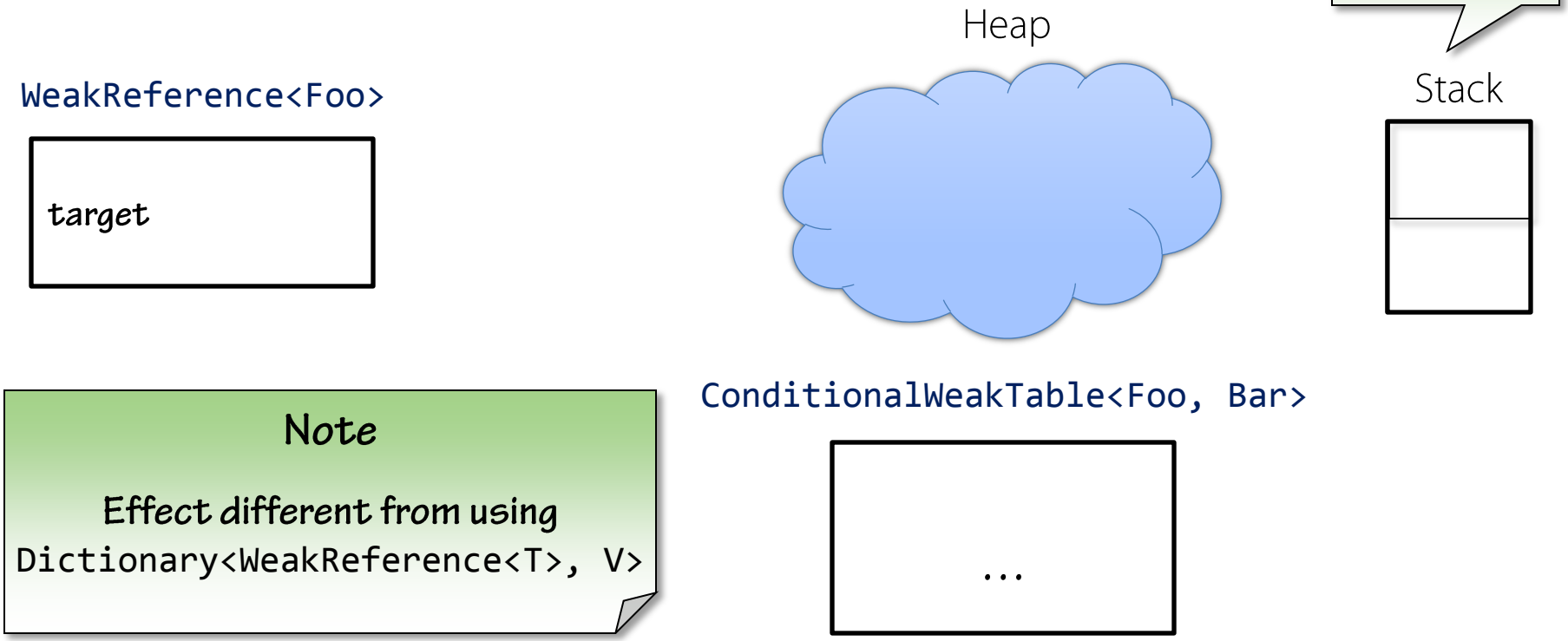
ConditionalWeakTable<TKey, TValue>

- **Object lifetimes**
  - Strong references are the default
  - Weak references
    - Targets of WeakReference and WeakReference<T>
    - Keys in ConditionalWeakTable<TKey, TValue>



ConditionalWeakTable<TKey, TValue>

- **Object lifetimes**
  - Strong references are the default
  - Weak references
    - Targets of WeakReference and WeakReference<T>
    - Keys in ConditionalWeakTable<TKey, TValue>



Note

Effect different from using `Dictionary<WeakReference<T>, V>`

ConditionalWeakTable<TKey, TValue>

- **Building "extension properties"**
  - Use Get* and Set* extension methods
  - Associate data with object using ConditionalWeakTable

```
static class FooExtensions {
  private static ConditionalWeakTable<Foo, Hashtable> properties
          = new ConditionalWeakTable<Foo, Hashtable>();

  public static string GetBar(this Foo foo) {
    return (string)properties.GetOrCreateValue(foo)["Bar"];
  }

  public static void SetBar(this Foo foo, string value) {
    properties.GetOrCreateValue(foo)["Bar"] = value;
  }
}
```

MethodImplAttribute

- **Optimization**
  - NoInlining
    - Disables inlining of current method
    - Useful to guarantee presence of the method on the stack
    - Examples in the BCL:
      - Methods dealing with StackCrawlMark
      - GC.KeepAlive used by JIT
  - AggressiveInlining (.NET 4.5)
    - Hints the JIT that inlining is beneficial
    - Used in a handful of places in the BCL
  - NoOptimization
    - Disables JIT (and NGEN) optimizations for the method
    - Useful to demonstrate JIT behavior
    - Sometimes used for tools that require deep inspection (e.g. Parallel Debugger)

MethodImplAttribute

- **Synchronized**
  - Similar to synchronized methods in Java
  - Behavior:
    - Locks on the type for static members
    - Locks on the instance for instance members
  - Can be harmful:
    - Others can lock on these objects too
    - Still useful for private classes

- **Other flags**
  - ForwardRef used for type forwarders
  - InternalCall used within the BCL
  - Unmanaged to indicate the method contains native code

IndexerNameAttribute

- **The mysterious Item property**
  - Used by indexers in C#
  - Interoperability for languages that don't have indexers
  - Getter and setter with indexer parameters
    - Remember: properties are just metadata

```
class Vector {
   public double this[int x] {
      get { … } set { … }
   }

   // CS0102: The type 'Vector' already
   // contains a definition for 'Item'
   public int Item {
      get; set;
   }
}
```

To fix, apply IndexerNameAttribute

get_Item and set_Item

Caller Info Attributes

- **Introduced in C# 5.0**
  - Embed compile-time information in the code
    - No runtime reflection or StackTrace calls
  - __FILE__ and __LINE__ from C/C++
    - Relies on macros
  - Design using optional parameters (added in C# 4.0)
    - [CallerMemberName]
      - Method, property, event name
      - Internal names for constructors, destructor, operators
    - [CallerLineNumber]
    - [CallerFilePath]

```
static class Logger {
  public static void Log(string message,
                         [CallerMemberName] string caller = null,
                         [CallerFilePath] string file = null,
                         [CallerLineNumber] int line = -1) { … }
}
```

Implementing INotifyPropertyChanged

- **No more magic strings that don't survive refactoring**
  - Less boilerplate code

```
class Person : INotifyPropertyChanged {
  private string name;

  public string Name { get { return name; }
                       set { OnPropertyChanged(ref name, value); } }

  public event PropertyChangedEventHandler PropertyChanged;

  protected virtual void OnPropertyChanged<T>(ref T field, T value,
                                [CallerMemberName] string caller = null) {
    if (!EqualityComparer<T>.Default.Equals(field, value)) {
      field = value;
      var changed = propertyChanged;
      if (changed != null)
        changed(this, new PropertyChangedEventArgs(caller));
    }
  }
}
```

No boilerplate code here!

Summary

- **ConditionalWeakTable<TKey, TValue>**
  - "Extend" objects at runtime
  - Add to the WeakReference toolset
- **MethodImplOptions**
  - Disable JIT optimizations to see what's going on
  - Consider when AggressiveInlining makes sense
- **IndexerNameAttribute**
- **Caller Info Attributes**
  - Useful for logging libraries, INotifyPropertyChanged
  - Beware of brittleness due to compile time info usage