C# 4.0
“dynamic”

Oliver Sturm
http://www.oliversturm.com

Outline

- **The "dynamic" keyword**
  - Where it can be used and where it can't
- **How it works**
  - Clever C# compiler trickery
- **A common misconception regarding Reflection**
- **Interfacing with dynamic worlds, demonstrated**
  - Excel Automation
  - IronPython

The “dynamic” keyword

- Working with compiled .NET types is easy in C#
- Type information is available through Reflection
  - It is possible to make runtime calls dynamically
- When a decision about calls and call targets is made at runtime, we talk about “dynamic dispatch”, or “late binding”
- Other language platforms have their own dynamic dispatch techniques (Python, Ruby, but also Automation, Web Services, …)
- “dynamic” builds a bridge to the Dynamic Language Runtime (DLR)
- The DLR unifies different dynamic dispatch approaches

First Impressions

```
dynamic i = 42;
dynamic s = "Hi there";
i.DoSomethingImpossible();
…
foreach (dynamic thing in things)
  DoFall(thing);
```

- "dynamic" is used in place of a type
- "dynamic" does NOT correspond to any type
- Runtime types of variables are similar to what you'd expect when using "var"
- Member access on variables declared "dynamic" is deferred to runtime

Things that don't work

- Deriving from "dynamic"
- Implementing IEnumerable<dynamic>
- Extension methods for "dynamic"

- Remember: "dynamic" is not a type

How It Works – Static Typing

Original code:

```
Test test = new Test( );
test.DoSomething( );
```

Decompiled by Reflector:

```
new Test().DoSomething();
```

How It Works – Dynamic Typing

Original code:

```
dynamic test2 = new Test( );
test2.DoSomething( );
```

Decompiled by Reflector:

```
object test2 = new Test();
if (<Main>o__sc0.<>p__s1 == null) {
  <Main>o__sc0.<>p__s1 =
    CallSite<Action<CallSite, object>>.Create(
      Binder.InvokeMember(
        CSharpBinderFlags.ResultDiscarded, "DoSomething",
          null, typeof(Program), new CSharpArgumentInfo[]
          { CSharpArgumentInfo.Create(CSharpArgumentInfoFlags.None,
            null) }));
}
<Main>o__sc0.<>p__s1.Target(<Main>o__sc0.<>p__s1, test2);
```

The Reflection Misconception

- Dynamic dispatch through the DLR is quite fast
- The idea is obvious to replace mechanisms like Reflection with DLR dynamic dispatch
- The APIs offered by the DLR don't make this easy
- The "dynamic" keyword makes it easy because the compiler does the work
- If information relevant to the dispatch is not available at compile time, "dynamic" can't help

Summary

- "dynamic" keyword builds a bridge to the DLR
- The DLR knows how to dispatch into different target systems
- A unified way of interfacing with dynamic platforms from C#

References

- C# 4.0 Language Specification: http://osturm.me/cs40spec