C# to D

Adam Wilson
Director of Software Development
Prospective Software

Topics Covered

- Syntax
- Properties
- Namespaces
- Generics

- LINQ
- Collections

- My Favorite Thing I can’t do in D

Random Syntax Notes

C#

- var
- readonly
- string
- [Attributes]
- params int[]

D

- auto
- immutable
- wstring
- @(Attributes)
- int[] …

Properties

- C#

```
public int data { get; protected set; }
public int data { get { return m_data; } set { m_data = value; } }
```

- D

```
@property int data() { return m_data; }
@property int data(int value) { return m_data = value; }
```

Converting C# Properties to D

- Relatively straight-forward for standard C# properties
- Can be somewhat time consuming
- Find-and-Replace not useful
- Well suited to automated tooling

Differences in Properties

- C# cannot take the address of a property
- Resolves D's awkward syntax when property is delegate
  - C# Example:
    public Action<int> DelegateProp { get; } //declaration
    DelegateProp(1); //call the delegate
  - D Example:
    @property delegate(int) DelegateProp() { return m_data; }
    DelegateProp()(1); //UFCS requires the first parentheses
- The difference is semantic, but highly confusing.
- Have yet to encounter real-world need to take address of a property

Namespaces

- C#

```
namespace System { }
namespace System.Text { }
```

- D

```
module System;
module Text;
```

Systemic Differences

- C# Namespaces
  - Composable
  - Single namespace can span many source files
  - Structure is determined by the namespace itself
  - Used for logical source organization
- D Modules
  - Not Composable
  - Name can only exist in a single file
  - Uses directory structure to determine namespace
  - No intrinsic organization capability

Converting C# Namespaces to D

- Create the corresponding directory structure
- Move source all C# source into a single file
- Place single source file in correct directory
- Set correct module name in source file
- Example:
  - namespace System.Text.RegularExpressions { }
  - System directory
    - Text directory
      - RegularExpressions.d

Observations on Modules in D

- Modules are NOT Namespaces!
- Modules are more analogous to C# Nested Types
- A Module with the same name as the parent directory is an error
- Leaf namespace is really a source file
- Leaf-namespace-as-source-file encourages source files of unusual size
- Lack of composability also encourages source files of unusual size
- Lack of composability prevents programmer from extending module
- Example of Namespace Composability in C#: System.Utilities
  - https://github.com/prospectivesoftware/systemutilities

More Observations on Modules in D

- Public imports reduce typing but also reduce import granularity
- Public import example in System directory:

```
module Imports;
public import Text;
public import Collections;
public import IO;
public import Net;
public import ServiceModel;
```

Potential Workaround for Public Imports

- Create an Imports.d for every package in the library
- System
  - Imports.d
  - Text
    - Imports.d
    - RegularExpressions.d
  - Collections
    - Imports.d
    - Generic.d
    - Concurrent.d
    - Specialized.d

DIP15 & DIP16

- Resolves most limitations of D Modules in relation to C# Namespaces
- Renders previously demonstrated workaround unneeded
- Still not composable
- Credit to Martin Nowak and Andrei Alexandrescu
- Links: http://wiki.dlang.org/DIP15 - http://wiki.dlang.org/DIP16
- Please support these DIP's!

Generics

- C# Generics allow the users of the code to specify one or more types.
- Less flexible than D's template system
- Generics can be considered a subset of D's template system
- D's template constraints are considerably more expressive than C#'s
- C# Generic Constraints:
  - struct
  - class
  - new()
  - Base Class
  - Interface
  - Another generic type

C# Generics Example

```
public class SomeBaseClass {}
public class SomeOtherClass : SomeBaseClass {}

public class SomeClass<T> where T : SomeBaseClass
{
        private List<T> values;
        void Add(T input) { values.Add(T); }
}
```

D Generics Example

```
public class SomeBaseClass {}
public class SomeOtherClass : SomeBaseClass {}

class SomeClass(T : SomeBaseClass)
{
        private T[] values;
        public void Add(T input) { values ~= input; }
}
```

LINQ Set Operations

C#

- Distinct
- Except
- Intersect
- Union

D

- uniq
- setDifference
- setIntersection
- setUnion

LINQ Aggregation Operations

C#

- Aggregate
- Average
- Count
- LongCount
- Min
- Max
- Sum

D

- reduce
- count
- min
- max

LINQ Sorting Operations

C#

- OrderBy
- OrderByDescending
- ThenBy
- ThenByDescending
- Reverse

D

- sort
- sort
- reverse

Other LINQ Operations

C#

- Where
- GroupBy
- Contains
- Repeat
- SequenceEqual
- Concat

D

- filter
- group
- canFind
- fill
- equal
- joiner

LINQ vs. std.algorithm

- This list may not be complete
- Neither has everything available in the other, for example:
- There is no equivalent to map() in LINQ, use SelectMany and GroupBy
- Most common operations exist in both
- Equivalent operation names not always obvious
- Internal semantics of equivalent functions may differ
- Verify that your output is equivalent!

D Containers for C# Programmers

C#

- T[]
- Dictionary<TKey, TValue>
- List<T>
- *Not Available*
- LinkedList<T>
- SortedSet<TValue> (.NET 4.0+)
  SortedDictionary<TKey, TValue>

D

- T[]
- TValue[TKey]
- Array(T)
- SList(T)
- DList(T)
- RedBlackTree(TValue)

D Containers That Can

- Phobos provides equivalent functionality for basic C# collections operations
- Operations: Add/Insert/Remove/Clear/Empty/Length/Capacity
- Both support LINQ/std.algorithm operations.

D Containers That Can’t

- Std.container is missing many of the containers available in C#:
- Generic: Queue, Stack, HashSet, ObservableCollection
- Concurrent: ConcurrentDictionary, ConcurrentQueue, ConcurrentStack, ConcurrentBag, BlockingCollection
- Synchronized: SynchronizedCollection, SynchronizedKeyedCollection
- Read-Only: ReadOnlyCollection, ReadOnlyDictionary, ReadOnlyObservableCollection, SynchronizedReadOnlyCollection

My Favorite Thing I can't do in D

```
public string Name { get { return GetValue(NameProperty); } set
{ SetValue(NameProperty, value); } }

public static readonly DeltaProperty<string> NameProperty =
DeltaProperty<string>.Register("Name", typeof(Customer), default(string), null);
```

My Favorite Thing I can't do in D

```
public abstract class DeltaObject
{
        private ConcurrentDictionary<HashID, object> values;
        public T GetValue<T>(DeltaProperty<T> de)
        {
                object value;
                return values.TryGetValue(de.ID, out value) == false ? de.DefaultValue : (T)value;
        }
        public void SetValue<T>(DeltaProperty<T> de, T value)
        {
                object temp;
                if (EqualityComparer<T>.Default.Equals(value, de.DefaultValue))  values.TryRemove(de.ID, out temp);
                else temp = values.AddOrUpdate(de.ID, value, (p, v) => value);
        }
}
```

My Favorite Thing I can't do in D

- A Pattern for Concurrently Mutable Data
- Why can't D do this?
  - Uses the 'default' keyword
  - Uses ConcurrentDictionary and ConcurrentQueue
  - Uses SHA2 hashing functions
- Implementation is simple, but relies on library and language features not present in D.
- Source: https://github.com/prospectivesoftware/netpathsdk

.NET Namespace Comparison

- The following namespaces have no analog in Phobos:
- System.Collections (Except System.Collections.Generics)
- System.IO
- System.Text (Except Regular Expressions)
- System.Data
- System.Timers
- System.Printing

A Call to Arms

- Much of the C# language can be ported to D
- Significant portions of the Base Class Library cannot be ported to D
- In practical examples, software that relies on even basic .NET BCL types cannot be easily ported
- However, there are many cases where D meets and even exceeds the capabilities of C#
- Containers are one of D's major weaknesses compared to C# and .NET
- How can we push containers forward today?