Documentation and Functional Specification

February 20, 2008

1. XML Documentation in C#

2. Functional Specification

Inline XML Documentation

- Visual Studio/Mono can generate XML documentation from comments in source files.
- Generated XML can be turned into web pages, or used by other tools
- Support for custom tags is potentially useful for third party tools.

Example: Inline XML Documentation

```
namespace Geometry
{
  /// <summary>
  /// This text explains the <c>Point</c> class.
  /// </summary>
  class Point
  {
    public int x;
    public int y;

    /// <summary>
    /// Moves the point
    /// </summary>
    /// <param name="dx">Amount to move</param>
    public void moveX(int dx){ x+=dx; }
  }
}
```

Example: Generated XML

```xml
<?xml version="1.0"?>
<doc>
  <assembly>
    <name>pointFile</name>
  </assembly>
  <members>
    <member name="T:Geometry.Point">
      <summary>
      This text explains the <c>Point</c> class.
      </summary>
    </member>
    <member name="M:Geometry.Point.moveX(System.Int32)">
      <summary>
      Moves the point
      </summary>
      <param name="dx">Amount to move</param>
    </member>
  </members>
</doc>
```

Basics of XML Comments

- All XML comments must be on /// lines
- XML comments must proceed either
  - Type declarations: classes, delegates, interfaces
  - Member declarations: fields, events, properties, and methods
- Members without xml comments are omitted from the documentation
- XML comments can't be used
  - in method bodies
  - on namespaces . . .

Standard Tags

- <summary> General information about a member</summary>
- <value> Describes property value</summary>
- <param name="x">Description of method parameter x</param>
- <returns>Description of method results</param>
- <exception cref="*name*">Describes an exception that may be thrown</exception>
- <seealso cref="*name*">A cross reference</seealso>...

Visual studio has special support for some tags.

- <summary>— text shown by Intellisense
- <param>— compiler checks parameter names are correct
- <exception>— compiler checks that the exception type exists

Member name decoration

All members names are decorated with their full names, types of their arguments, and a one-character label.

Recall:

```
public void moveX(int dx){ x+=dx; }
```

⇝

```
<member name="M:Geometry.Point.moveX(System.Int32)">
```

XML Documentation Character Labels

Label	**Meaning**
T	Type: class, interface, struct, enum, delegate
F	Field
P	Property
M	Method
E	Event
N	Namespace (C# can't document namespaces, but can reference them.)
!	Error

Summary: Inline XML Documentation

- Benefits:

- Drawbacks:

Summary: Inline XML Documentation

- Benefits:
  - Code and documentation in one place—kept is sync
  - Standard tags facilitate tool use
- Drawbacks:

Summary: Inline XML Documentation

- Benefits:
  - Code and documentation in one place—kept is sync
  - Standard tags facilitate tool use
- Drawbacks:
  - Documentation can overwhelm code in source files
  - Hard to internationalize documentation—should translation team need to edit source files?

Summary: Inline XML Documentation

- Benefits:
  - Code and documentation in one place—kept is sync
  - Standard tags facilitate tool use
- Drawbacks:
  - Documentation can overwhelm code in source files
  - Hard to internationalize documentation—should translation team need to edit source files?
- Alternative programs (e.g. monodoc) try to provide the best of both worlds.

Summary: Inline XML Documentation

- Benefits:
  - Code and documentation in one place—kept is sync
  - Standard tags facilitate tool use
- Drawbacks:
  - Documentation can overwhelm code in source files
  - Hard to internationalize documentation—should translation team need to edit source files?
- Alternative programs (e.g. monodoc) try to provide the best of both worlds.
- No required documentation system in this class.

1 XML Documentation in C#

2 Functional Specification

Useful documentation

Documentation should fully specify what code does.
Questions documentation should answer:

- What state does an object model?
- What are method pre- and post-conditions?
- What can cause exceptions, and which exceptions?
- What assumptions and invariants are used by the implementation?

Our approach: Document a program's behavior using well-defined clauses that discuss different aspects of specification.

Functional specification and abstraction.

- Implementation should be hidden from clients.
- Maintainers need all the details.

- Principle: Document public things using an abstract *specification state* to describe program behavior.
- Principle: Document private things using both the specification and *concrete state* of program.

Documenting classes and interfaces.

Classes and interfaces should be described generally, and define the associate specification state.

Example

```
// Instances of Point represent
// the geometric object.
// State: A point p in R^2
class Point{ ... }
```

Documenting private fields

Private members define the concrete state of a class. Document their invariants, and define an *abstraction function* defining how concrete and abstract states are related.

```
// Polar radius of the point.
// Invariant: r >= 0.
private double r;

// Polar angle of the point.
// Invariant: 0 <= theta < 0
private double theta;

// Abstraction Function:
// State p = (r*cos(theta), r*sin(theta))
```

Documenting public members

Public members should be described in terms of the abstract state.

```
// this.X is p's X component
public double X{ get {r * sin(theta);}
                 set {...} }
```

Documenting methods

Write method specifications that describe the pre- and post-conditions of the method, including possible side-effects and exceptions.

```
// distance(q) returns the distance
//   between p and q.
double distance(Point q)

// rotate(d) effects this by rotating p about
//   the origin by d radians
// Requires: -pi < d <= pi
void rotate(double d).
```

Specification clauses

State Abstract state of a class.

Abstraction Function Relates abstract and concrete states.

Invariants Constraints on public or private fields or members. Invariants must hold after any constructors executes.

Checks Method pre-condition. Method promises to throw an exception when violated.

Requires Method pre-condition. Method may or may not throw an exception when violated.

Throws Method post-condition. Explains a possible thrown exception.

Returns Method post-condition describing ordinary return values.