1 In-code Documentation

Documentation is a very important part of writing programs, since it is most unlikely that you will be writing really obvious code. Moreover, what seems obvious at the point of writing may be mystifying months later to the author and to others. Documentation serves several purposes:

1. Communicate what the code should be doing.
2. Highlight big insights essential for the code.
3. Highlight possible conflicts and/or areas where the code could be changed later.

The essential point is that coding is a journey in problem-solving, and proper documentation is an aid in understanding the solution and the journey that lead to it. Documentation is most often a mixture of in-code documentation and accompanying documents. Here, we will focus on in-code documentation which arguably causes problems in multi-language environments and run the risk of bloating code.

F# has two different syntaxes for comments. Comments can be block comments:

Listing 1.1: Block comments.

```
(*<any text>*)
```

The comment text `(<any text>)` can be any text and is stilled parsed by F# as keywords and basic types, implying that `(* a comment (* in a comment *) *)` and `(* "*)" *)` are valid comments, while `(* " *)` is invalid.

Alternatively, comments may also be line comments,

Listing 1.2: Line comments.

```
//<any text>
```

where the comment text ends after the first newline.

The F# compiler has an option for generating *Extensible Markup Language* (*XML*) files from scripts using the C# documentation comments tags[1]. The XML documentation starts with a triple-slash `///`, i.e., a lineComment and a slash, which serve as comments for the code construct that follows immediately after. XML consists of tags which always appear in pairs, e.g., the tag "tag" would look like `<tag> ... </tag>`. F# accept any tags, but recommends those listed in Table 1.1. If no tags are used,

Tag	Description
`<c>`	Set text in a code-font.
`<code>`	Set one or more lines in code-font.
`<example>`	Set as an example.
`<exception>`	Describe the exceptions a function can throw.
`<list>`	Create a list or table.
`<para>`	Set text as a paragraph.
`<param>`	Describe a parameter for a function or constructor.
`<paramref>`	Identify that a word is a parameter name.
`<permission>`	Document the accessibility of a member.
`<remarks>`	Further describe a function.
`<returns>`	Describe the return value of a function.
`<see>`	Set as link to other functions.
`<seealso>`	Generate a See Also entry.
`<summary>`	Main description of a function or value.
`<typeparam>`	Describe a type parameter for a generic type or method.
`<typeparamref>`	Identify that a word is a type parameter name.
`<value>`	Describe a value.

Table 1.1: Recommended XML tags for documentation comments, from ECMA-334 3rd Edition, Annex E, Section 2.

then it is automatically assumed to be a `<summary>`. An example of a documented script is shown in Listing 1.3. is:

[1]For specification of C# documentations comments see ECMA-334: `http://www.ecma-international.org/publications/files/ECMA-ST/Ecma-334.pdf`

**Listing 1.3 commentExample.fsx:
Code with XML comments.**

```
/// The discriminant of a quadratic equation with
/// parameters a, b, and c
let discriminant a b c = b ** 2.0 - 4.0 * a * c

/// <summary>Find x when 0 = ax^2+bx+c.</summary>
/// <remarks>Negative discriminants are not
    checked.</remarks>
/// <example>
///   The following code:
///   <code>
///     let a = 1.0
///     let b = 0.0
///     let c = -1.0
///     let xp = (solution a b c +1.0)
///     printfn "0=%.1fx^2+%.1fx+%.1f => x_+=%.1f" a b c xp
///   </code>
///   prints <c>0=1.0x^2+0.0x+-1.0 => x_+=0.7</c>.
/// </example>
/// <param name="a">Quadratic coefficient.</param>
/// <param name="b">Linear coefficient.</param>
/// <param name="c">Constant coefficient.</param>
/// <param name="sgn">+1 or -1 determines the
    solution.</param>
/// <returns>The solution to x.</returns>
let solution a b c sgn =
  let d = discriminant a b c
  (-b + sgn * sqrt d) / (2.0 * a)

let a = 1.0
let b = 0.0
let c = -1.0
let xp = (solution a b c +1.0)
printfn "0 = %.1fx^2 + %.1fx + %.1f => x_+ = %.1f" a b c xp
```

Mono's `fsharpc` command may be used to extract the comments into an XML file, as demonstrated in Listing 1.4.

Listing 1.4, Converting in-code comments to XML.

```
$ fsharpc --doc:commentExample.xml commentExample.fsx
F# Compiler for F# 4.0 (Open Source Edition)
Freely distributed under the Apache 2.0 Open Source License
```

This results in an XML file with the content shown in Listing 1.5.

Listing 1.5, An XML file generated by `fsharpc`.

```
<?xml version="1.0" encoding="utf-8"?>
<doc>
<assembly><name>commentExample</name></assembly>
<members>
<member name="M:CommentExample.solution(System.Double,
  System.Double,System.Double,System.Double)">
 <summary>Find x when 0 = ax^2+bx+c.</summary>
 <remarks>Negative discriminants are not checked.</remarks>
 <example>
   The following code:
   <code>
     let a = 1.0
     let b = 0.0
     let c = -1.0
     let xp = (solution a b c +1.0)
     printfn "0 = %.1fx^2 + %.1fx + %.1f => x_+ = %.1f" a b
   c xp
   </code>
   prints <c>0 = 1.0x^2 + 0.0x + -1.0 => x_+ = 0.7</c> to
  the console.
 </example>
 <param name="a">Quadratic coefficient.</param>
 <param name="b">Linear coefficient.</param>
 <param name="c">Constant coefficient.</param>
 <param name="sgn">+1 or -1 determines the solution.</param
  >
 <returns>The solution to x.</returns>
</member>
<member name="M:CommentExample.discriminant(System.Double,
  System.Double,System.Double)">
<summary>
 The discriminant of a quadratic equation with parameters a
  , b, and c
</summary>
</member>
</members>
</doc>
```

The extracted XML is written in C# type by convention, since F# is part of the Mono and .Net framework that may be used by any of the languages using Assemblies. Besides the XML inserted in the script, the XML has added the `<?xml ...>` header, `<doc>`, `<assembly>`, `<members>`, and `<member>` tags. The header and the `<doc>` tag are standards for XML. The extracted XML is geared towards documenting big libraries of codes and thus highlights the structured programming organisation, see **????**, and `<assembly>`, `<members>`, and `<member>` are indications for where the functions belong in the hierarchy. As an example, the prefix `M:CommentExample.` indicates that the method is in the namespace `commentExample`, which in this case is the name of the file. Furthermore, the function type

```
val solution : a:float->b:float->c:float->sgn:float->float
```

is in the XML documentation

M:CommentExample.solution(System.Double,System.Double,System.Double,System.Double),

which is the C# equivalent.

An accompanying program in the Mono suite is `mdoc`, whose primary use is to perform a syntax analysis of an assembly and generate a scaffold XML structure for an accompanying document. With the `-i` flag, it is further possible to include the in-code comments as initial descriptions in the XML. The XML may be updated gracefully by `mdoc` as the code develops, without destroying manually entered documentation in the accompanying documentation. Finally, the XML may be exported to HTML.

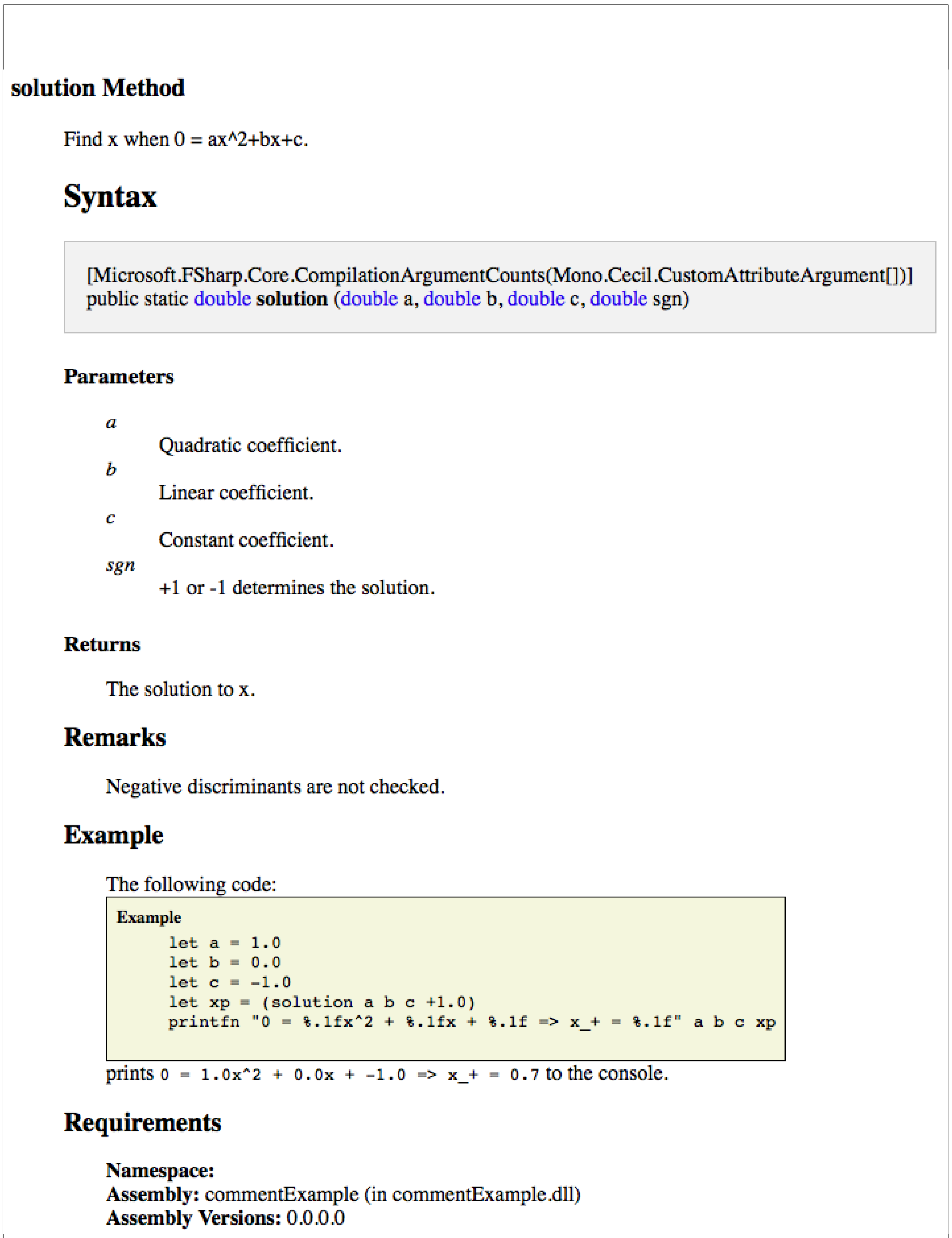
The primary use of the `mdoc` command is to analyze compiled code and generate an empty XML structure with placeholders to describe functions, values, and variables. This structure can be updated and edited as the program develops, and the edited XML files can be exported to *Hyper Text Markup Language* (*HTML*) files and viewed in any browser. Using the console, all of this is accomplished by the procedure shown in Listing 1.6, and the result is shown in Figure 1.1.

Listing 1.6, Converting an XML file to HTML.

```
$ mdoc update -o commentExample -i commentExample.xml
  commentExample.exe
New Type: CommentExample
Member Added: public static double determinant (double a,
  double b, double c);
Member Added: public static double solution (double a,
  double b, double c, double sgn);
Member Added: public static double a { get; }
Member Added: public static double b { get; }
Member Added: public static double c { get; }
Member Added: public static double xp { get; }
Namespace Directory Created:
New Namespace File:
Members Added: 6, Members Deleted: 0
$ mdoc export-html -out commentExampleHTML commentExample
.CommentExample
```

A full description of how to use `mdoc` is found here[2].

[2] `http://www.mono-project.com/docs/tools+libraries/tools/monodoc/generating-documentation/`



solution Method

Find x when 0 = ax^2+bx+c.

Syntax

```
[Microsoft.FSharp.Core.CompilationArgumentCounts(Mono.Cecil.CustomAttributeArgument[])]
public static double solution (double a, double b, double c, double sgn)
```

Parameters

a
Quadratic coefficient.
b
Linear coefficient.
c
Constant coefficient.
sgn
+1 or -1 determines the solution.

Returns

The solution to x.

Remarks

Negative discriminants are not checked.

Example

The following code:

Example

```
let a = 1.0
let b = 0.0
let c = -1.0
let xp = (solution a b c +1.0)
printfn "0 = %.1fx^2 + %.1fx + %.1f => x_+ = %.1f" a b c xp
```

prints `0 = 1.0x^2 + 0.0x + -1.0 => x_+ = 0.7` to the console.

Requirements

Namespace:
Assembly: commentExample (in commentExample.dll)
Assembly Versions: 0.0.0.0

Figure 1.1: Part of the HTML documentation as produced by `mdoc` and viewed in a browser.