Code

You might have heard more than once that **you should write self-documenting code**. This doesn't mean that you should never comment your code.

There are two types of code comments, implementation comments and documentation comments.

Implementation Comments

They are used for internal documentation, and are intended for anyone who may need to maintain the code in the future, including your future self.

There can be single line and multi-line comments (e.g., C# Comments). Comments are human-readable and not executed, thus ignored by the compiler. So you could potentially add as many as you want.

Now, the use of these comments is often considered a code smell. If you need to clarify your code, that may mean the code is too complex. So you should work towards the removal of the clarification by making the code simpler, easier to read, and understand. Still, these comments can be useful to give overviews of the code, or provide additional context information that is not available in the code itself.

Examples of useful comments:

- Single line comment in C# that explains **why** that piece of code is there (from a private method in System.Text.Json.JsonSerializer):

```
// For performance, avoid obtaining actual byte count unless
memory usage is higher than the threshold.
Span<byte> utf8 = json.Length <=
(ArrayPoolMaxSizeBeforeUsingNormalAlloc /
JsonConstants.MaxExpansionFactorWhileTranscoding) ? ...
```

- Multi-line comment in C# that provides **additional context** (from a private method in System.Text.Json.Utf8JsonReader):

```
// Transcoding from UTF-16 to UTF-8 will change the length by
somewhere between 1x and 3x.
// Un-escaping the token value will at most shrink its length
by 6x.
// There is no point incurring the transcoding/un-
escaping/comparing cost if:
// - The token value is smaller than charTextLength
// - The token value needs to be transcoded AND unescaped and
it is more than 6x larger than charTextLength
//      - For an ASCII UTF-16 characters, transcoding = 1x,
escaping = 6x => 6x factor
//      - For non-ASCII UTF-16 characters within the BMP,
transcoding = 2-3x, but they are represented as a single
escaped hex value, \uXXXX => 6x factor
//      - For non-ASCII UTF-16 characters outside of the BMP,
transcoding = 4x, but the surrogate pair (2 characters) are
represented by 16 bytes \uXXXX\uXXXX => 6x factor
// - The token value needs to be transcoded, but NOT escaped
and it is more than 3x larger than charTextLength
//      - For an ASCII UTF-16 characters, transcoding = 1x,
//      - For non-ASCII UTF-16 characters within the BMP,
transcoding = 2-3x,
//      - For non-ASCII UTF-16 characters outside of the BMP,
transcoding = 2x, (surrogate pairs - 2 characters transcode to
4 UTF-8 bytes)

if (sourceLength < charTextLength
    || sourceLength / (_stringHasEscaping ?
JsonConstants.MaxExpansionFactorWhileEscaping :
JsonConstants.MaxExpansionFactorWhileTranscoding) >
```

```
charTextLength)
{
```

Documentation Comments

Doc comments are a special kind of comment, added above the definition of any user-defined type or member, and are intended for anyone who may need to use those types or members in their own code.

If, for example, you are building a library or framework, doc comments can be used to generate their documentation. This documentation should serve as API specification, and/or programming guide.

Doc comments won't be included by the compiler in the final executable, as with single and multi-line comments.

Example of a doc comment in C# (from Deserialize method in System.Text.Json.JsonSerializer):

```
/// <summary>
/// Parse the text representing a single JSON value into a
<typeparamref name="TValue"/>.
/// </summary>
/// <returns>A <typeparamref name="TValue"/> representation of
the JSON value.</returns>
/// <param name="json">JSON text to parse.</param>
/// <param name="options">Options to control the behavior
during parsing.</param>
/// <exception cref="System.ArgumentNullException">
/// <paramref name="json"/> is <see langword="null"/>.
/// </exception>
/// <exception cref="JsonException">
/// The JSON is invalid.
///
/// -or-
///
```

```
/// <typeparamref name="TValue" /> is not compatible with the
JSON.
///
/// -or-
///
/// There is remaining data in the string beyond a single JSON
value.</exception>
/// <exception cref="NotSupportedException">
/// There is no compatible <see
cref="System.Text.Json.Serialization.JsonConverter"/>
/// for <typeparamref name="TValue"/> or its serializable
members.
/// </exception>
/// <remarks>Using a <see cref="string"/> is not as efficient
as using the
/// UTF-8 methods since the implementation natively uses UTF-8.
/// </remarks>
[RequiresUnreferencedCode(SerializationUnreferencedCodeMessage)]
public static TValue? Deserialize<TValue>(string json,
JsonSerializerOptions? options = null)
{
```

In **C#**, doc comments can be processed by the compiler to generate XML documentation files. These files can be distributed alongside your libraries so that Visual Studio and other IDEs can use IntelliSense to show quick information about types or members. Additionally, these files can be run through tools like DocFx to generate API reference websites.

More information:

- Recommended XML tags for C# documentation comments.

In other languages, you may require external tools. For example, **Java** doc comments can be processed by Javadoc tool to generate HTML documentation files.

More information:

- How to Write Doc Comments for the Javadoc Tool
- Javadoc Tool

Last update: August 22, 2024