Table of Contents

[Introduction 1](#_Toc527814517)

[1) Notation explanation: 1](#_Toc527814518)

[2) Tag Documentation table 1](#_Toc527814519)

[<|qc|> 1](#_Toc527814520)

[<native> 1](#_Toc527814521)

[<$> 1](#_Toc527814522)

[<namespace> 1](#_Toc527814523)

[<class> 1](#_Toc527814524)

[<struct> 1](#_Toc527814525)

[<interface> 1](#_Toc527814526)

[<property> 2](#_Toc527814527)

[<indexer> 2](#_Toc527814528)

[<function> 2](#_Toc527814529)

[<for> 2](#_Toc527814530)

[<foreach> 2](#_Toc527814531)

[<while> 2](#_Toc527814532)

[<do> 2](#_Toc527814533)

[<if> 2](#_Toc527814534)

[<else if> 2](#_Toc527814535)

[<else> 2](#_Toc527814536)

[<forever> 2](#_Toc527814537)

[<unchecked><unsafe> 2](#_Toc527814538)

[<?any-other-tags?> 2](#_Toc527814539)

[Physical examples 3](#_Toc527814540)

[Classes 3](#_Toc527814541)

[Structs 3](#_Toc527814542)

[Functions 3](#_Toc527814543)

[Interfaces & Properties 3](#_Toc527814544)

# Introduction

## Notation explanation:

* 1. !xyz! means that “xyz” is a language keyword & must be inserted as is
  2. .!xyz!. is like !xyz!, but it’s optional.
  3. ?xyz? means that “xyz” has a contextual meaning & should be used and inserted accordingly.
  4. .?xyx?. is like ?xyz?, but it’s optional.
  5. .!xyz!.!abc!. means that “xyz” & “abc” are language keywords & also that either of then must be inserted as is.
  6. NOTE: This notation is not a language feature but is only used in this documentation.

## Tag Documentation table

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Tag | Tag Short hand | Reason for Selected Shorthand | Description | Usage |
| <|qc|> | - | - | In case if parsing fails insert this tag at the global scope, it is needed if there is code outside of a block, so this global block was invented. | <|qc|>  ?the-rest-of-all-the-qc-code?  </> |
| <native> | - | - | Inserts the code between this block as is. | <native>  ?native-code?  </> |
| <$> | - | - | To represent comments. | <$>  ?content?  </$> |
| <namespace> | - | - | To represent namespaces. | <namespace>  …namespace elements…  </> |
| <class> | <?> | In C# classes by default are nullable i.e. they may have a value or they may not i.e. you can’t always be sure that they have a value, without checking, hence “?”. | To represent classes. | <class> ?name?  ……………  </> |
| <C> | The abbreviation of class, “C”. |
| <struct> | <!> | In C# structs are by default ***not*** nullable, hence “!”. | To represent structs. | <struct> ?name?  ……………  </> |
| <S> | The abbreviation of struct, “S”. |
| <interface> | <^> | Interfaces have to have a class or struct inheriting them ***up*** the inheritance hierarchy, hence “^”. | To represent interfaces. | <interface> ?name?  <property> ?name? of ?type?  .!get!.  .!set!.  </>  …more properties…  ?name?(?argument-list?) .?!gives! type?.  …more functions…  <indexer> [?argument-list?] of ?type?  .!get!.  .!set!.  </>  …more indexers…  </> |
| <I> | The abbreviation of interface, “I”. |
| <property> | <=> | Properties are set-able, hence “=”. | To represent properties. | <property> ?name? of ?type?  .<get>  ?get-implementation?  </>.  .<set>  ?set-implementation?  </>.  </> |
| <prop>  <P> | The abbreviation of property, “prop” & “P”. |
| <indexer> | <[]> | Objects are indexed with “[” & “]”, hence “[]” | Used in indexable objects. | <indexer> [?argument-list?] of ?type?  <get>  ?get-implementation?  </>  .<set>  set-implementation?  </>  </> |
| <idxr> | The abbreviation of indexer, “idxr”. |
| <function> | <()> | Functions are called with “(” & “)”, hence “()” | To represent functions. | <function> ?name? (?argument-list?) .?!gives! type?.  ?implementation?  </> |
| <func>  <F> | The abbreviation of function, “func” & “F”. |
| <for> | - | - | Used as the for loop. | <for> ?initialization?; ?condition?; ?increment?  ?body?  </> |
| <foreach> | <in> | The foreach loop loops through all the elements ***in*** a collection, hence “in”. | Used as the foreach loop. | <foreach> ?variable-declaration? in ?collection?  ?body?  </> |
| <while> | <W> | The abbreviation of while, “W”. | Used as the while loop. | <while> ?condition?  ?body?  </> |
| <do> | - | - | Used as the do…while loop. | <do>  ?body?  </> while ?condition? |
| <if> | - | - | Used as the if…elif…else construct | <if> ?condition?  ?body?  </>  <elif> ?condition?  ?body?  </>  <else>  ?body?  </> |
| <else if> | <elif> | “elif” has been famously used in many languages in-place of “else if”. |
| <else> | - | - |
| <forever> | - | - | Used as an infinite loop e.g.  *while( true)* or  *for( ;; )* | <forever>  ?body-ideally-containing-a-*break*-statement?  </> |
| <unchecked><unsafe> | - | - | Represents the same constructs as in C#. | <.!unckecked!.!unsafe!.>  ?body?  </> |
| <?any-other-tags?> | - | - | <?any-other-tag?>  ?body?  </> |

# Physical examples

|  |  |  |
| --- | --- | --- |
|  | Shorthand QC | Equivalent C# |
| Classes | <C> Vector2  <$>  The X component of the vector.  </$>  public float x  <$>  The Y component of the vector.  </$>  public float y  <$>  Constructor  </$>  <F> (float x = 0f, float y = 0f) gives Vector2  this.x = x  this.y = y  </>  </> | class Vector2  {  /\*  The X component of the vector.  \*/  public float x;  /\*  The Y component of the vector.  \*/  public float y;  /\*  Constructor  \*/  Vector2 (float x = 0f, float y = 0f)  {  this.x = x;  this.y = y;  }  } |
| Structs | <S> Vector2  <$>  The X component of the vector.  </$>  public float x  <$>  The Y component of the vector.  </$>  public float y  <$>  Constructor  </$>  <F> (float x = 0f, float y = 0f) gives Vector2  this.x = x  this.y = y  </>  </> | struct Vector2  {  /\*  The X component of the vector.  \*/  public float x;  /\*  The Y component of the vector.  \*/  public float y;  /\*  Constructor  \*/  Vector2 (float x = 0f, float y = 0f)  {  this.x = x;  this.y = y;  }  } |
| Functions | <F> ProcessValues( params object[] vals ) gives object  object giveVal = null  <$>  Do computation with *vals* and set *giveVal*.  </$>  give giveVal  </>  <F> ProcessValuesAndOutput( params object[] vals )  <$>  Assumed output function: *Print(string)*  </$>  Print( ProcessValues( vals ).ToString() )  </> | object ProcessValues( params object[] vals )  {  object giveVal = null;  /\*  Do computation with *vals* and set *giveVal*.  \*/  return giveVal;  }  void ProcessValuesAndOutput( params object[] vals )  {  /\*  Assumed output function: *void* *Print(string)*  \*/  Print( ProcessValues( vals ).ToString() );  } |
| Interfaces & Properties | <I> IList{{T}}: ICollection{{T}}, IEnumerable{{T}}, IEnumerable  <[]> [ int index ] of T  get  set  </>  IndexOf(T item) gives int  Insert(int index, T item)  RemoveAt(int index)  </>  <I> ICollection{{T}}: IEnumerable{{T}}, IEnumerable  <P> Count of int  get  </>  <P> IsReadOnly of bool  get  </>  Add(T item)  Clear()  Contains(T item) gives bool  Remove(T item) gives bool  </> | interface IList<T>: ICollection<T>, IEnumerable<T>, IEnumerable  {  T this[ int index ]  {  get;  set;  }  int IndexOf(T item);  void Insert(int index, T item);  void RemoveAt(int index);  }  interface ICollection<T>: IEnumerable<T>, IEnumerable  {  int Count  {  get;  }  bool IsReadOnly  {  get;  }  void Add(T item);  void Clear();  bool Contains(T item);  bool Remove(T item);  } |