[Dafny Standard Library Documentation](https://dafny-lang.github.io/dafny)

Dafny Standard Library Style Guide

* Naming Convention
  + Method Prefix
* Code Layout
  + Braces
  + Imports
* Indentation and Line Breaks
  + Tabs or Spaces?
  + Maximum Character Limit
  + Newlines
  + Functions, Methods, Predicates, and Lemmas
* Content Conventions
  + Order
  + Predicates
* Things to Avoid
  + Parentheses
  + [Whitespace](https://dafny-lang.github.io/dafny/StyleGuide/Style-Guide.html#whitespace)
    - Type Declaration
    - Function, Method, Predicate, and Lemma Declaration
* Recommendations
  + Externs
  + Things to Consider

This style guide provides coding conventions for the Dafny Standard Library code.

*This documentation is still in progress. Please feel free to add more suggestions.*

Naming Convention

Any **variables** are named with camelCase.

var minValue := 1;

var cipherMessage := "Hello World";

Any **classes**, **modules**, **datatypes**, and **newtypes** are named with PascalCase.

Module MyModule {

...

}

Any **lemmas**, **predicates**, **functions** and **methods** are named with snake\_case.

method find\_index(arr: seq<int>, k: int)

...

The **lemma** keyword indicates a ghost method used for proof purposes. Any **lemma** names should be prefixed with lemma\_.

lemma lemma\_value\_is\_in\_index(arr: seq<int>, k: int)

...

Any static or global **constants** are named with UPPERCASE\_WITH\_UNDERSCORES.

static const MONTHS\_IN\_A\_YEAR := 12

Method Prefix

Avoid redundant names when variables or methods are in a class/module.

class Integer {

// The following method converts the given integer

// to a string.

//

// this method name can be simplified to to\_string()

// so that the method call is Integer.to\_string(i)

// instead of Integer.integer\_to\_string(i).

// YES

method to\_string(i: int) returns (s: string)

...

// NO

method integer\_to\_string(i: int) returns (s: string)

...

}

Code Layout

Braces

Opening braces go on the same line by default.

module M {

...

method met() {

...

}

}

In case the method (or function, lemma, etc) signature is too long to fit in one line, or in case the signature has at least one specification clause, the opening brace goes on a new line.

module M {

...

method met(i: int) returns (j: int)

requires i % 2 == 0

ensures j > 10

{

...

}

}

This applies to every scope: module, class, predicate, if, while, and more.

Imports

By default, import modules without opening them.

import Coffee

...

However, if some members of a module are used very frequently, import it using opened:

import opened Donut

...

When a file uses two modules and both of them define a method of the same name, do not import them opened.

import MyModule

import YourModule

...

method my\_method() {

my\_module.foo();

your\_module.foo();

}

In this case, if you want to shorten the module name, import it with a shorthand name.

import M = MyModuleWithACumbersomeName

import Y = YourModuleWithACumbersomeName

...

method my\_method() {

M.foo();

Y.foo();

}

Common imports, such as StandardLibrary and Native, should be grouped together, followed by custom module imports with a blank line in-between.

import opened StandardLibrary

import opened Native

import opened Donut

import Coffee

Although not required, it’s recommended to keep the order of imports and includes alphabetical, except when it makes more sense to group them logically.

Indentation and Line Breaks

Tabs or Spaces?

Spaces are preferred over tabs. Tabs should only be used to remain consistent with existing code containing tabs.

Use 2 spaces for each indentation.

Maximum Character Limit

Although there is no strict requirement, it is generally recommended to have a maximum of 120 characters per line.

Newlines

Use newlines between sequential **functions**, **methods**, **predicates**, and **lemmas** to increase readability.

End each file with a newline.

Functions, Methods, Predicates, and Lemmas

Every Dafny method has the following signature.

method {:<attributes>} method\_name(param1: Type, param2: Type) returns (ret: Type)

requires P()

modifies param2

ensures Q()

decreases param1

When possible, put method\_name and the returns statement on the same line, as the keyword returns is distinct from other method specification clauses, such as requires, modifies, ensures, and decreases, which should appear in this order. Each method specification clause should be on a separate line, indented.

In case the Method signature is too long, we can break it down.

method {:<attributes>} method\_name(param1: Type, param2: Type,

param3: Type, param4: Type, param5: Type)

returns (ret1: Type, ret2: Type, ret3: Type, ret4: Type,

ret5: Type)

requires P1()

requires P2()

requires P3()

modifies param2

modifies param3

ensures Q1()

ensures Q2()

decreases param1

Multiple requires or ensures can be combined into one:

requires

&& P1()

&& P2()

&& P3()

The same rules apply to function, predicate, and lemma definitions.

Content Conventions

Order

Functions, predicates, and methods within a file should be sorted topologically, meaning that everything method M depends on should be above M in the file.

function my\_function(a: int): int{

...

}

method my\_method(i: int) {

...

return my\_function(i);

}

Predicates

Predicates should be used instead of functions that return a Boolean value.

// YES

predicate foo() {

...

}

// NO

Function foo():bool {

...

}

Things to Avoid

Parentheses

In many cases, Dafny does not require parentheses around expressions. Here are some examples.

* If-Else-While Statements

// YES

var i := 1;

while i < 10 {

...

if 1 < i {

...

}

...

}

// NO

var i := 1;

while (i < 10) {

...

if (1 < i) {

...

}

...

}

* Statements That Take Expression Arguments

// YES

assert x < 100;

print x;

// NO

assert(x < 100);

print(x);

* Simple Boolean/Arithmetic Expressions

// YES

method collatz(num: nat)

decreases \*

{

var n := num;

while 1 < n

decreases \*

{

n := if n % 2 == 0 then n / 2 else n \* 3 + 1;

}

}

// NO

method collatz(num: nat)

decreases \*

{

var n := num;

while (1 < n) // unnecessary parentheses

decreases \*

{

n := if ((n % 2) == 0) then (n / 2) else ((n \* 3) + 1); // unnecessary parentheses

}

}

Whitespace

Avoid unnecessary whitespace inside expressions.

Type Declaration

A type declaration should have a form of variableName: variableType.

// YES

const one: int := 1

class {:extern} Util {

var {:extern} Exception: System.String

}

// NO

const one : int := 1 // unnecessary whitespace

class {:extern} Util {

var {:extern} Exception : System.String // unnecessary whitespace

}

If the type can be inferred by Dafny, leave it out, unless you think it provides useful documentation in the program. So, constant one above is better declared as

const one := 1

Function, Method, Predicate, and Lemma Declaration

The function, method, predicate, and lemma definitions should have the form function\_name(parameterName: parameterType, ...).

// YES

function method foo<int>(i: int): int

// NO

function method foo<int> (i : int) : int // unnecessary whitespace

Avoid too little or too much whitespace that reduces the overall readability.

// YES

lemma lemma\_my\_lemma<A, B>(x: seq<seq<A>>, y: B) {

...

}

// NO

lemma lemma\_my\_lemma <A,B> ( x : seq<seq<A>> , y :B){

...

}

Recommendations

This section describes a few recommendations that can help make code more readable and easy to follow, although not strictly enforced.

Externs

Try to name them the same in Dafny and the target language (e.g. C#, Java, etc) whenever possible, so that in Dafny we only have to write {:extern}, not {:extern "<name>"}.

Things to Consider

Ask these questions before designing / implementing a program in Dafny.

* Is this variable name / function name X a good name?
* Does it make sense that this method M is in module X? Shouldn’t it be in module Y instead?
* Does the definition X belong to the file Y.dfy?
* Is X.dfy a good filename?