March 2023

PowerSharp Programming Language

Reference Manual

Version 0.3 – PowerMess

Developed by Gabriel Margarido

Resume
PowerSharp is a strong, explicit or inferred and static typed programming language, that compiles to Go source/executable binary. It aims make easy Go development for normal people, it's syntax was inspired in C, C++, Solidity, Ruby, Crystal, Go and Lua programming languages.

To run PowerSharp correctly you should have installed before it: Node.js 14+, NPM 8+, GNU Make and Golang.

You can download them through this links:
Node.js and NPM: www.nodejs.org
Golang: www.go.dev
GNU Make for Windows: https://gnuwin32.sourceforge.net/packages/make.htm

Gabriel Margarido,
March 2023

Installing compiler from sources:

1. Install these softwares first: GNU Make, Node.js 12+, NPM 8+ and Golang.
(And also **Git Bash** if you are running in Windows.)

Installing Go v1.20.2 from sources (UNIX/Windows)

(Enter inside the unzipped directory)

B. Installing Go from sources (macOS/Linux):
```
cd go/src && ./all.bash
```

C. Installing Go from sources (Windows):
```
cd go\src && start all
```

Installing PowerSharp Linux

Install Go lang: https://go.dev/dl/

A. On Ubuntu or Debian you can run:
```
sudo apt install nodejs npm make -y
sudo make all install test
```

Installing PowerSharp MacOS X

Install Node.js and NPM: https://www.nodejs.org
Install Go lang: https://go.dev/dl/

C. (unzip and enter inside the downloaded directory,
next run the following commands)

```
sudo make all install test
```

Installing PowerSharp Windows

Install Node.js and NPM: https://www.nodejs.org
Install GNU Make: https://gnuwin32.sourceforge.net/packages/make.htm
Install Go lang: https://go.dev/dl/

(unzip and enter inside the downloaded directory,
next run the following commands)

C. Then, run inside the unzipped directory
```
start all
```

D. Or you can run directly the Makefile
```
make microsoft-win64
```

They're gonna be installed on UNIX systems at:
```
/usr/local/bin/powerc
/usr/local/powerc-sample
```

And for Microsoft Windows inside:
```
dist-win64/powerc.exe
dist-win64/powerc-sample
```

4. To compile a source file
```
powerc -new myproject
```

4. To compile a source file
```
cd myproject
powerc main.pwr -o main
```

You can uninstall the compiler by running:
```
sudo make remove
```

These are all existing datatypes in PowerSharp:

Datatype	Description
`int int8 int16 int32 int64`	Signed Integer (negative or positive) value
`uint uint8 uint16 uint32 uint64`	Unsigned (positive) integer value
`rune`	Integer 32-bit value with unicode for symbols
`byte`	Alias for unsigned integer 8-bit
`float32 float64`	Decimal positive or negative real value
`complex64 complex128`	Float 64-bit/128-bit for imaginary values
`string`	String value
`bool`	Boolean (true or false) / 0 or 1

Variable declaration:

```
x = 53
int x = 53

x = -53
int8 x = -53
int16 x = -53
int32 x = -53
int64 x = -53


uint x = 53

uint8 x = 53
uint16 x = 53
uint32 x = 53
uint64 x = 53
byte x = 53


y = 7.52
float32 y = 7.52
float64 y = 7.52

z = “Hello world”
string z = “Hello world”


w = true
w = false

bool w = true
bool w = false
```

How to assign an undefined value:

```
string s = nil
r = nil          → Not recommended


int i = nil
r = nil          → Not recommended


float32 f = nil
f = nil          → Not recommended


bool b = nil
b = nil          → Not recommended
```

Variable with error handling declaration:

A. *How we do in PowerSharp*

```
@error err
file = os.Open(“filename.txt”)
```

B. *How we do in Go*

```
f, err := os.Open("filename.ext")
if err != nil {
    log.Fatal(err)
}
```

Variable reassignment:

```
x := 5
y := 0.5
z := “Bye world”
w := true        x := false
```

Errors *with variable reassignment (Strong-typed):*

```
x := 5.6                  → Error! Trying to cast integer to float
y := 2                    → Error! Trying to cast float to integer
z := 0                    → Error! Trying to cast string to int
w := “Hello world”        → Error! Trying to cast bool to string
```

Arrays declaration:

```
int  a[6] = (-0.5, 5.4, -332.45, -1.5, 4, 15)
uint a[6] = (0, 5, 3, 1, 4, 15)
float32 b[6] = (0.5, 5.23, 3.43, 1.15, 4.02, 15.43)
string  c[4] = (“Julia”, “Maria”, “Clara”, “Miriam”)
```

```
int  a[] = (-0.5, 5.4, -332.45, -1.5, 4, 15)
uint a[] = (0, 5, 3, 1, 4, 15)
float32 b[] = (0.5, 5.23, 3.43, 1.15, 4.02, 15.43)
string  c[] = (“Julia”, “Maria”, “Clara”, “Miriam”)
```

Showing messages on the screen:

```
print(“Hello world\n”)
puts(“Hello world\n”)
```

Getting user data:

- *First way (most recommended):*

```
string x = nil
gets(x)
```

- *Second way (less recommended):*
  `gets(&x)`

If-Conditional

```
if (condition) do
      ...
elsif (condition) do
      ...
else
      ...
end
```

While Loop

```
while (condition) do
      ...
end
```

Repetition loops

```
for iterator 0 to 5 do
      ...
end


5 times do
      ...
end
```

Human-readable operators: `is  isnot  and  or`
Machine-readable operators: `==  !=  and  or`

Writing mathematical expressions on the screen

```
float x = 42+b+z*k+(1/2+45/4)+(456/4)
puts(x)
```

Private function declaration:

```
def foo(a: string, b: int32, c: float32)  float32
      ...
      float32 d = (a+c)*b
      return d
end
```

```
def foo(a string, b int32, c float32)  float32
      ...
      float32 d = (a+c)*b
      return d
end
```

Function calling:

```
foo(45, 3.5, 420)
```

Public function declaration:

```
def Foo(a: string, b: int32, c: float32)  float32
      ...
      float32 d = (a+c)*b
      return d
end
```

```
def Foo(a string, b int32, c float32)  float32
      ...
      float32 d = (a+c)*b
      return d
end
```

Public function calling:

```
Foo(45, 3.5, 420)
```

Concatenating strings

```
a = “Hello”
b = “ ”

puts(a+b+”world”)
```

1. This section requires:

```
include “string_handling”
import “strings”
```

Getting length of string

```
string x = “Hello world”
int c = Length(x)

puts(c)
```

Putting string to lowercase

```
string x = “HELLO WORLD”
string c = Lowercase(x)

puts(c)
```

Putting string to uppercase

```
string x = “hello world”
string c = Uppercase(x)

puts(c)
```

2. This section requires:

```
include “env”
import “os”
```

Getting Environment Operating System

```
string x = GetOS()
puts(x)
```

Possible returns:

* -> *"windows"*
* -> *"linux"*
* -> *"darwin"*
* -> *"freebsd"*
* -> *"openbsd"*
* -> *"netbsd"*
* -> *"dragonfly"*
* -> *"solaris"*
* -> *"zos"*
* -> *"plan9"*
* -> *"hurd"*

```
* -> "illumos"
* -> "nacl"
* -> "js"
* -> "ios"
* -> "android"
* -> "aix"
```

Getting Environment CPU Architecture

```
string x = GetArch()
puts(x)
```

```
Possible returns:
* -> "386"
* -> "amd64"
* -> "amd64p32"
* -> "arm"
* -> "arm64"
* -> "arm64be"
* -> "loong64"
* -> "mips"
* -> "mips64"
* -> "mips64le"
* -> "mips64p32"
* -> "mips64p32le"
* -> "mipsle"
* -> "ppc"
* -> "ppc64"
* -> "ppc64le"
* -> "riscv"
* -> "riscv64"
* -> "s390"
* -> "s390x"
* -> "sparc"
* -> "sparc64"
* -> "wasm"
```

Running Shell Commands
Parameters → (command::string, isOutput::bool)

```
ShellCommand(“ls -a”, true)
ShellCommand(“ls -a”, false)
```

Comments in PowerSharp

```
@ (YOUR COMMENT HERE)
```

3. This section requires:

```
include “file”
import “io/ioutil”
```

Writing to text file

```
string x = “Hello world\n”
WriteToFile(“test.txt”, x)
```

Appending to text file

```
string x = “Hello world\n”
WriteToFile(“test.txt”, x)
```

Read from text file

```
string f = ReadFile(“test.txt”)
puts(f)
```

4. This section requires:

```
include “vector”
```

Removing from array by index (In this case: index 5, counting from 0)

```
int i[] = (34, 512, 32, 563, 256, 128)
ri = RemoveArrayFromIndex(i, 5)
puts(ri)
```

Removing from array by value (In this case: value 512 with index 1 counting from 0)

```
int i[] = (34, 512, 32, 563, 256, 128)
rA = RemoveArrayFromValue(i, 512)
puts(rA)
```

Appending to array (In this case: value 2048 with last index counting from 0)

```
int i[] = (34, 512, 32, 563, 256, 128)
xA = append(i, 2048)
puts(xA)
```

Main program structure

```
namespace org.yourname.projectname
package main

import “strings”
import “io/util”
import “os”
import “fmt”

include “string_handling”
include “file”
include “os“
include “vector”


@ (TODO CODE HERE)
def main()
      5 times do
            puts “Hello world”
      end
end
```

Import Go modules - *Import*

Modules are imported in default mode, like other Go modules, such as: `fmt` `os` `io/util`, and so on...

```
/usr/local/go/...

(.)
|
|_______mymodule
      |
      |____________ mymodule.go


import “mymodule”
```

Include Go libraries - *Include*

Library is read and stored inside the memory, then PowerC compiler writes to the file where it was called, unlike Go modules, they don’t be inside the Go global path or be a module.

```
(.)
|
|_______ power_modules
            |
            |_______mymodule
                  |
                  |____________ init.go


include “mymodule”
```

Call Go native functions
You can also call functions written
in Go inside PowerSharp source-code.

```
fmt.Println(“Hello world”)

fmt.Scanln(&x)
fmt.Println(x)
```

Declare Structures
Organize related data inside a single structure.

Declaration Syntax:

```
struct identifier do
      declare property int
      declare property float64
      declare property string
end
```

Instantiating Syntax:

```
identifier name = nil
```

Attributing Syntax:

```
name.property := value
```

Example:

```
struct vehicle do
      declare year int
      declare distance float64
      declare model string
end

vehicle honda = nil


honda.year := 2022
honda.distance := 512657.67
honda.model := “Civic”

puts(honda.year)
```

THE END