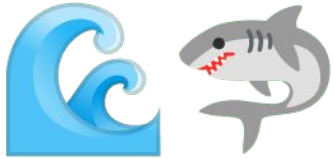

CShargs

argument parser

Evgenia Golubeva
Jan Kytka

Key use cases:

What we support:

Type safety (!)

Flags short/long: `cmd --option ≡ cmd -o`

Concat. flags: `cmd -Rf ≡ cmd -R -f`

Values: `cmd --key=value`

Variable length: `cmd --pos <x> <y> <z>`

Verbs: `git push`

User types: `--date=2021-04-15`

What we don’t support:

Range checks / any post parsing checks

Accepting dynamic parameters (parameters need to be known statically)

Overview: declarative approach with attributes

```
void Main(string[] args) {
    var arguments = new TimeArguments();
    arguments.Parse(args);

    // check version option
    if (arguments.Version) {
        Console.WriteLine("Version option present.");
    }

    if (arguments.Help) {
        // generate structured help, write it to console
        arguments.GenerateHelp(Console.Out);
    }

    // get parsed plain arguments
    var plainArgs = arguments.PlainArgs;
}
```

```
class TimeArgumentsParser : CShargs.Parser {

    // time  -V, --version
    [FlagOption("version", shortName: 'V', help: "Print version information.")]
    public bool Version { get; set; }

    // time --output=FILE
    [ValueOption("output", shortName:'o', required: false, help: "Do not send the results to stderr, but overwrite the
specified file.")]
    public string OutputFile { get; set; }

    // time --output=FILE -a
    [FlagOption("append", shortName:'a', useWith: nameof(OutputFile), help: "Do not overwrite but append.")]
    public bool Append { get; set; }

}
```

Overview: user defined types

Steps:

1. create custom type T
   with static Parse method:

```
public static T Parse(string stringValue);
// throws FormatException when parsing fails
```

2. use that type as an argument prop:

All basic types are handled by the exact same mechanism.

```
class NodeNumbers
{
    public int[] nodes;

    public static NodeNumbers Parse(string str)
    {
        var nodeNumbers = new NodeNumbers();
        ... (actual parsing goes here)
        return nodeNumbers;
    }
}

class NumactlOptions : CShargs.Parser {
    [ValueOption("interleave", shortName:'i', required: false)]
    public NodeNumbers Interleave { get; set; }
}
```

Overview: optional parameters vs non-nullable types

Options with `required: false` will have their **`default`** value unless stated otherwise

(Value types = zero)
(Reference types = null)

```
class OptionalArguments : CShargs.Parser {

    [ValueOption("amount1", required: false, help: "Amount. Defaults to 10")]
    public int WithDefault { get; set; } = 10;
    // required: false needs to be stated explicitly, int is not nullable

    [ValueOption("amount2", help: "Amount.")]
    public int? WithoutDefault { get; set; }
    // required: false is inferred automatically, because int? is nullable
}

...
{
    OptionalArguments args;

    args.WithDefault == 10 // when --amount1 not present
    args.WithoutDefault.HasValue == false // when --amount2 not present
}
```

Highlights:

- **Autogenerated error/help text**
- Easy way to report semantic errors
- Option groups (thanks to s12!)
- Aliases for multiple options

```
void Main(string[] args)
{
    var parser = new MyArguments();

    try {
        parser.Parse(args);

        if (parser.Help) {
            // help text is here:
            parser.GenerateHelp(Console.Out);
        }
    } catch (CShargs.ParsingException ex) {
        // error text is here
        Console.WriteLine(ex.Message);
    }
}
```

Help text includes:

- aliases
- default values (for non-required)
- option dependencies (use with)
- option groups

Highlights:

- Autogenerated error/help text
- **Easy way to report semantic errors**
- Option groups (thanks to s12!)
- Aliases for multiple options

Message from the exception will be included in the autogenerated error text.

(also possible to do without exceptions)

```
// user-defined type
class T
{
    public static T Parse(string value) {
        if (value is wrong format) {
            throw new System.FormatException("Bad format.");
        }
    }
}

// arguments object
class MyArguments : CShargs.Parser
{
    public int Amount { get; private set; }

    [CustomOption("amount", shortName: 'a', help: "Amount (0 to 5).")]
    public void ParseAmount(string value) {
        int Amount = int.Parse(value);

        if (Amount < 0 || Amount > 5) {
            throw new System.FormatException("Out of range.");
        }
    }
}
```

Highlights:

- Autogenerated error/help text
- Easy way to report semantic errors
- **Option groups**
- Aliases for multiple options

```
// command ( -w | -l )
[OptionGroup(required: true, nameof(Words), nameof(Lines))]
class CountArguments : Parser {

    [FlagOption("words", shortName: 'w')]
    bool Words { get; set; }

    [FlagOption("lines", shortName: 'l')]
    bool Lines { get; set; }
}
```

Can be now marked as not required, thanks s12!

Highlights:

- Autogenerated error/help text
- Easy way to report semantic errors
- Option groups (thanks to s12!)
- **Aliases for multiple options**

```
[AliasOption("a", nameof(Recursive), nameof(Force))]
class MyArguments : Parser {
    [FlagOption("recursive", shortName: 'r')]
    bool Recursive { get; set; }

    [FlagOption("force", shortName: 'f')]
    bool Force { get; set; }

    // option -a is now equivalent to -rf
}
```