

Building intuitive command-line interface in .NET

Alex Thissen

Cloud architect, Xebia | Xpirit

Level: Intermediate

Demo source code

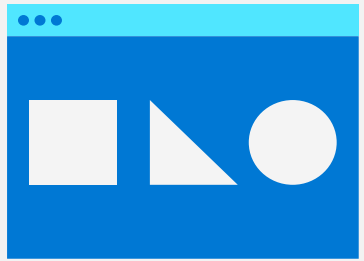
<https://github.com/alexthissen/modernclis>

<https://github.com/alexthissen/atarilynxemulator>



The rise of CLIs

Graphical User Interface



Command-line Interface



User experience

Rich UI and UX
Many visualizations

Cross-platform

Terminal-based or headless environments do not have GUI
Allows porting to Windows, Linux and MacOS

Efficiency

Fast and simple interaction model
Low level access in single line commands
Basic text-driven command line

Automation

Scriptable
Integration in CI/CD pipelines

Examples of CLIs and tools

CLIs

dotnet
az
docker
kubectl
tye
git
npm

Tools

curl
python

Interactive

netsh
mssql-cli
telnet
htprepl



```
Windows PowerShell
PS C:\> dotnet sln --help
sln
.NET modify solution file command

Usage:
dotnet [options] sln <SLN_FILE> [command]

Arguments:
<SLN_FILE> The solution file to operate on. If not specified, the command will search the current directory for one. [default: C:\]

Options:
-?, -h, --help Show help and usage information

Commands:
add <PROJECT_PATH> Add one or more projects to a solution file.
list List all projects in a solution file.
remove <PROJECT_PATH> Remove one or more projects from a solution file.

PS C:\> |
```


Demo

Exploring CLIs

Easter egg:

Windows Terminal Quake Mode:  + ` (backtick)

Command syntax

ale.exe

.\Zarlor.lnx

--magnification 3 -f --controller keyboard

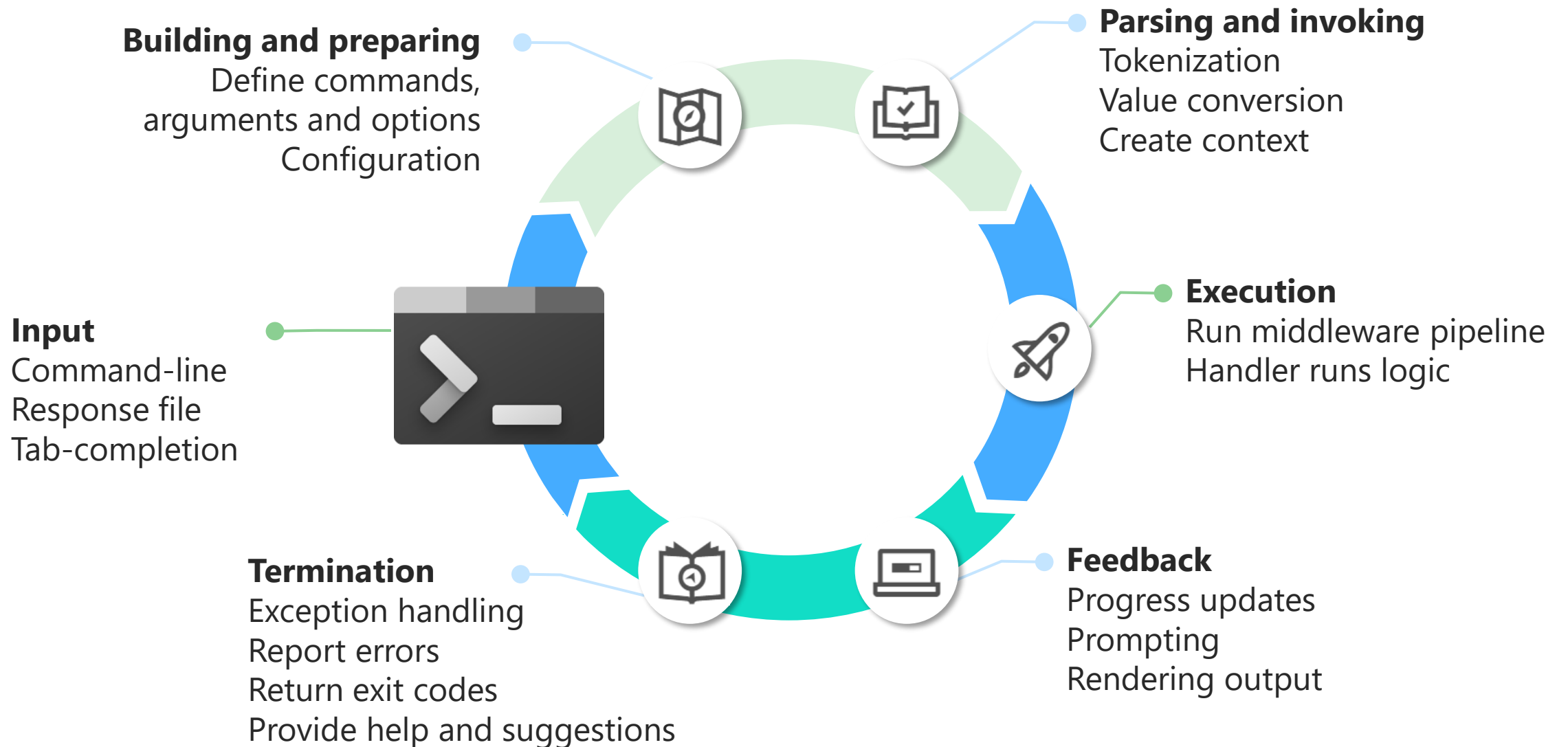
Arguments

- Specified without name
- Required values in order
- Arity (unary, binary, ...)

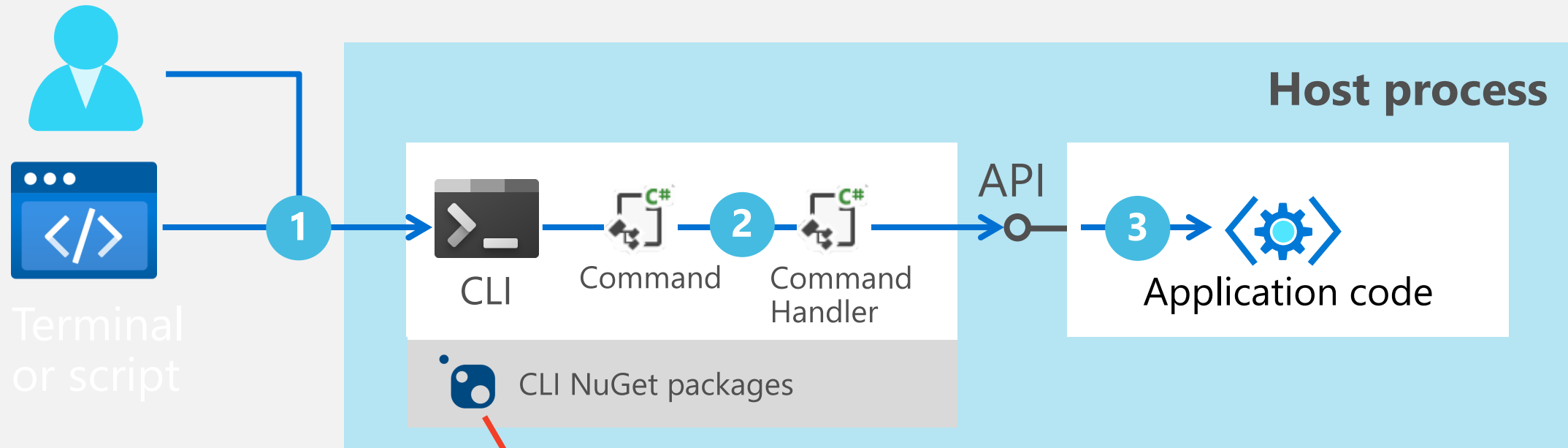
Options

- Key-value pairs
- Short and long form names
- Delimited by space, colon or equal sign
- Style: POSIX -- or Windows /
- Combine single char options
- Arity

Command-line lifecycle



Implementing CLIs in .NET

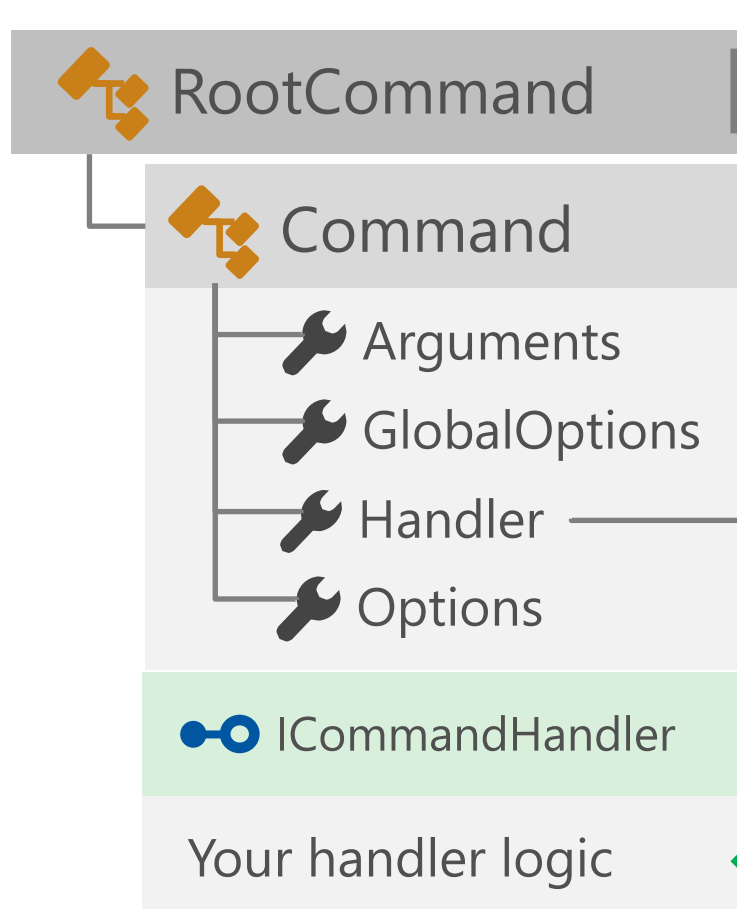


Recommended packages

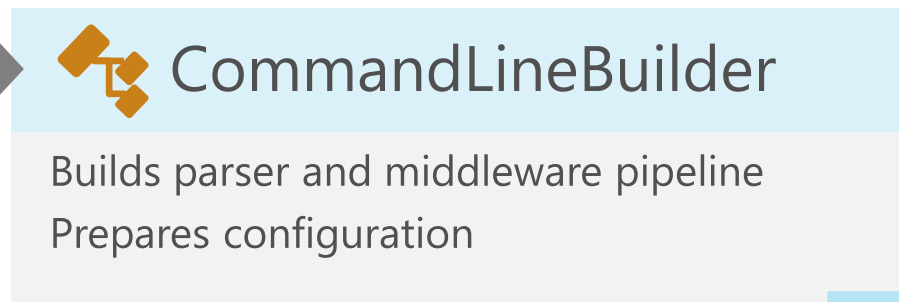
- **System.CommandLine**
- CliFx
- Spectre.Console.Cli
- CommandLineParser

System.CommandLine object model

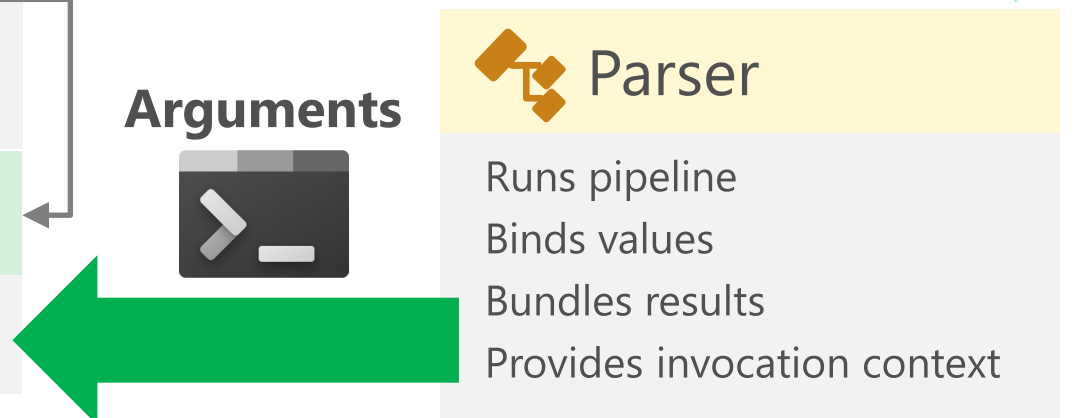
1 Prepare commands



2 Build command-line








3 Parse & Invoke



Symbols

Hierarchy of symbols
Name (and aliases for identifier symbols)

Other types

-  Option<T>
-  Argument<T>
-  IConsole
-  ISuggestionSource
-  IHelpBuilder

Demo

Building command-lines and parsers
Create commands, arguments and options

Command handling

Provide handler function per command

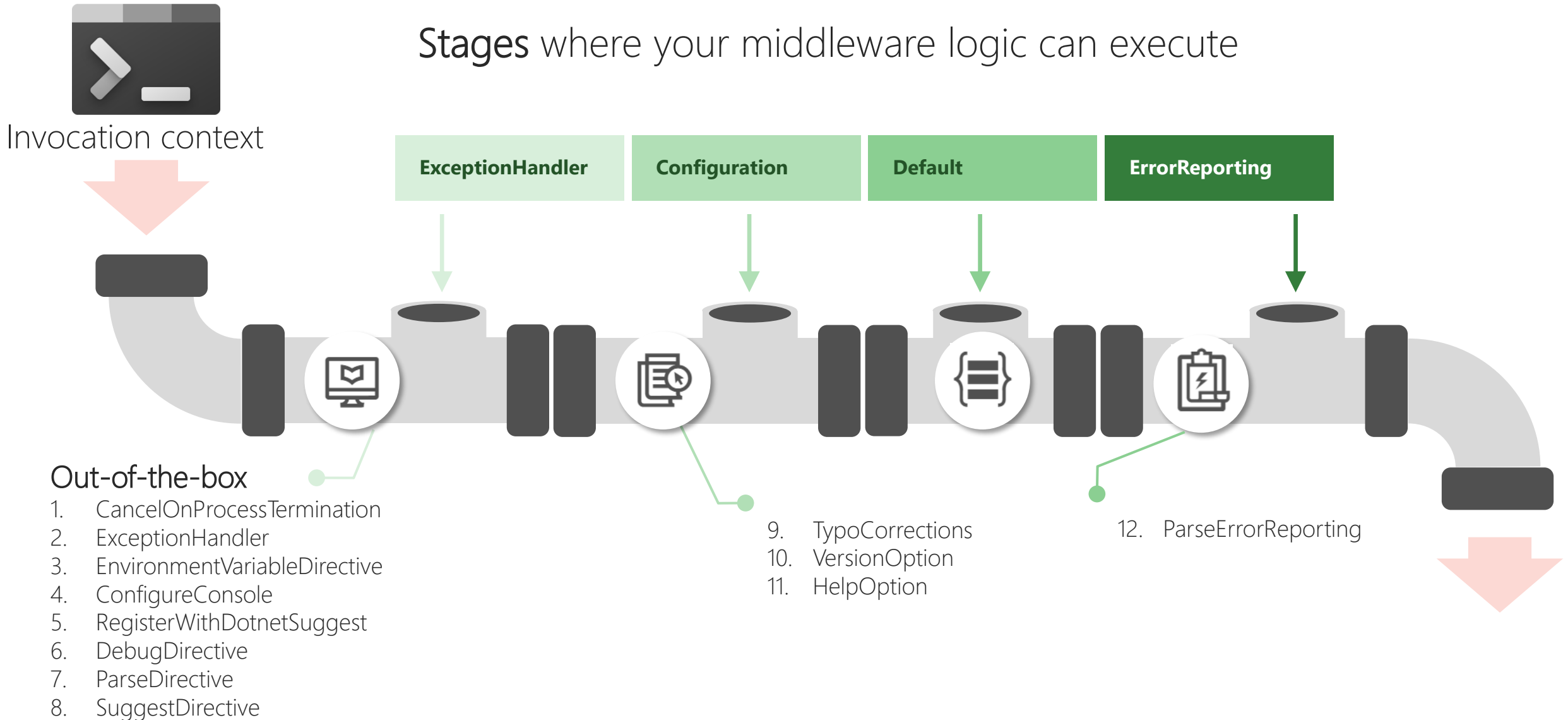
Create handler as lambda, delegate or separate function

Model binding no longer used by default (since beta 2)

```
rootCommand.SetHandler((InvocationContext context, int magnification, bool fullScreen) =>
{
    context.Console.Out.Write($"Magnification: {magnification} FullScreen: {fullScreen}");
}, magnificationOption, fullScreenOption
);
```



Middleware pipeline



Registering middleware

Extension methods on CommandLineBuilder

Resembles ASP.NET Core middleware

Various Use... methods

Prioritize with `MiddlewareOrder` enumeration

```
Parser parser = new CommandLineBuilder(rootCommand)
    .UseDefaults()
```

```
    .UseMiddleware(async (context, next) =>
    {
        // Execute your middleware logic here

        await next(context);
    }, MiddlewareOrder.ErrorReporting)
```

```
    .Build();
```

Order of registration

More than one custom middleware allowed

Beware of defaults

Production version might need to skip some default middleware (for example: directives)

Model binding

Uses `ModelBindingCommandHandler` implementation

 `System.CommandLine.NamingConventionBinder`

In-order named arguments

Named options

```
ale.exe .\Zarlor.lnx --full-screen -m 3 --controller keyboard
```

```
Run(CommandLineOptions gameOptions, bool fullScreen, ControllerType controller)
```

Boolean, integers, enumerations (case-insensitive)
Enumerable types (arrays, lists, ...)
String based constructors (`FileInfo`, `DirectoryInfo`, `Uri`)
Complex types (binding to properties or via constructor)

Case insensitive
Kebab casing supported
Ignores hyphens and other prefixes

Data types conversion

Special types are bound automatically

<code>IConsole</code>	Abstraction for out and err
<code>ParseResult</code>	All results of parsing tokens command-line
<code>CancellationToken</code>	Cancel async tasks
<code>BindingContext</code>	Captures converted typed values
<code>HelpBuilder</code>	Captures converted typed values
<code>InvocationContext</code>	Combines all of the above

```
Run(EmulatorClientOptions controller, BindingContext context, CancellationToken token)
```



**Simply add to
command handler signature**

Demo

Advanced scenarios

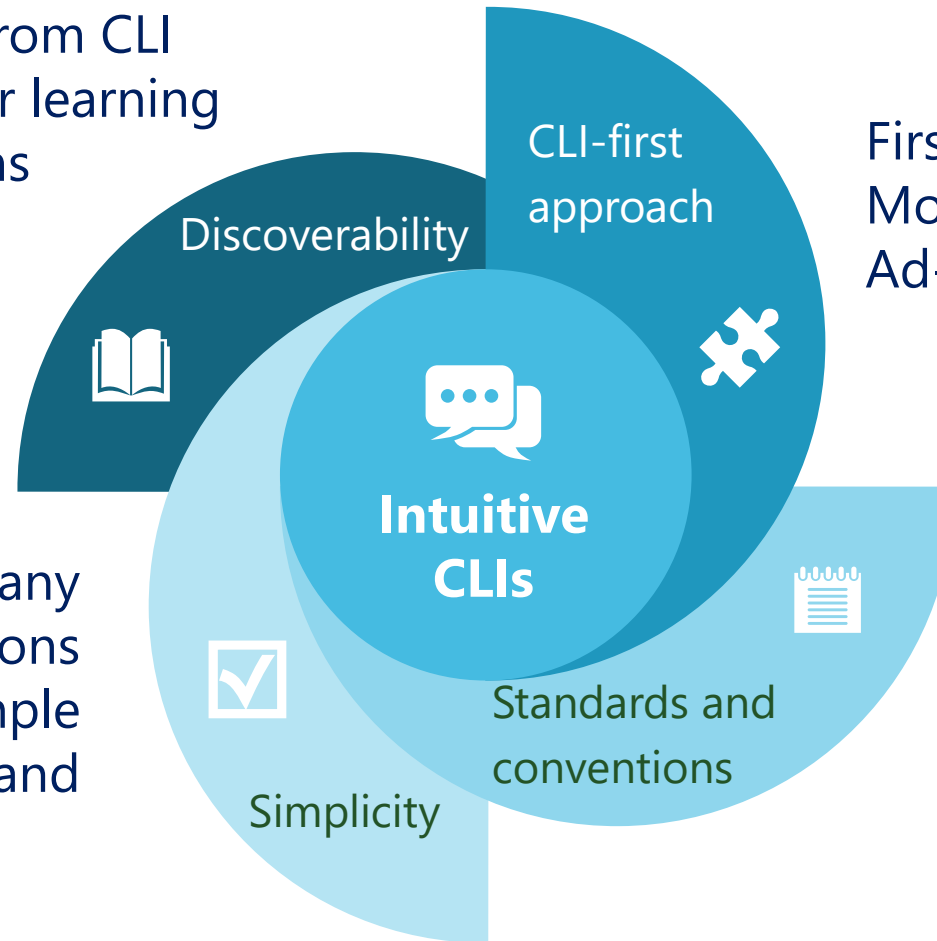
Subcommands and global options

Error handling and termination

Common patterns

Making it intuitive

Offer help and documentation from CLI
Defaults save time and better for learning
Suggestions and typo corrections



First client application is a CLI
More human-oriented:
Ad-hoc tasks and quick

Split a complex system into many
small functions
Assign each function to a simple
command

Familiar names of arguments
and options
Default to **stderr** and **stdout**

Help

Get contextual help for free

Description

Available names

Required and arity

Allowed values and default value

Provide custom help

Create your own help builder

Register.UseHelpBuilder<T**HelpBuilder**>

Single option or argument

Add or replace help sections

Reuse existing layouts



```
PS C:\> dotnet run -- --help
DemoCLI
  Atari Lynx Emulator

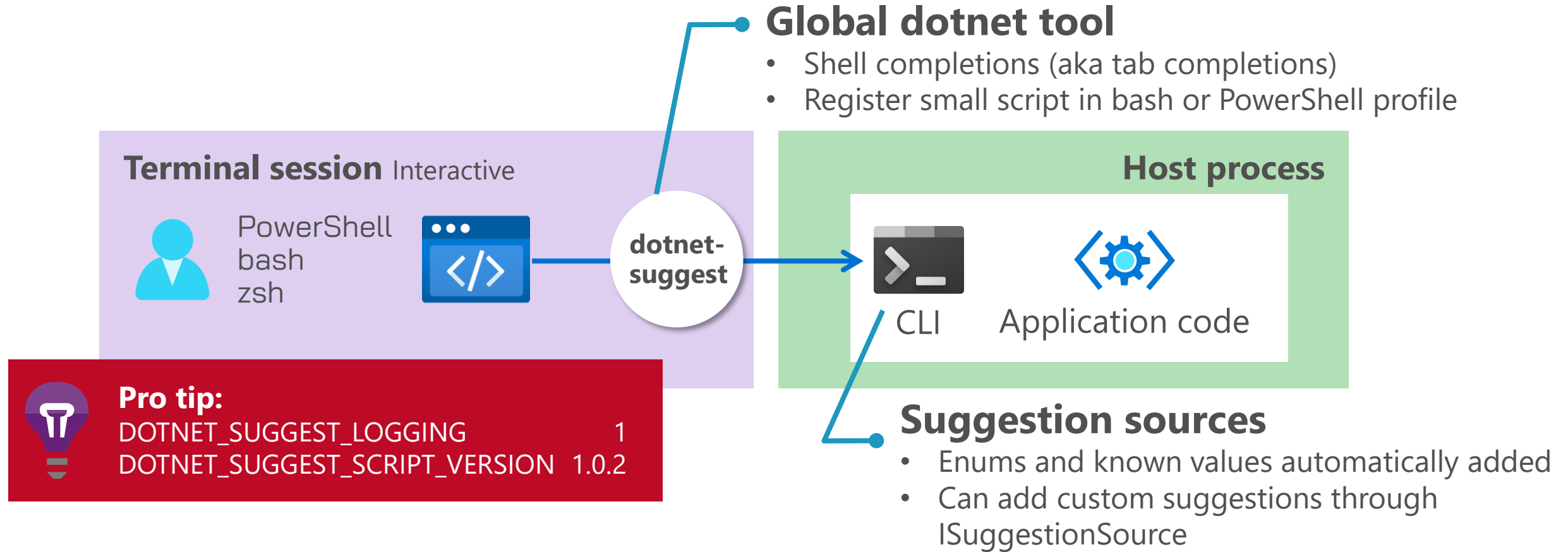
Usage:
  DemoCLI [options] <gamerom> [--] <additional arguments>...]]

Arguments:
  <gamerom>    Game ROM file

Options:
  -f, --fullscreen                Run full screen [default: False]
  -c, --controller <Gamepad|Keyboard> Type of controller to use [default: Keyboard]
  -m, --magnification <magnification> Magnification of screen [default: 4]
  --version                      Show version information
  -?, -h, --help                Show help and usage information

Additional Arguments:
  Arguments passed to the application that is being run.
```


Completions and typo corrections



```
Set-ExecutionPolicy -ExecutionPolicy RemoteSigned -Scope CurrentSession
dotnet tool install --global dotnet-suggest --version 1.1.327201
```

Debugging tips



Hint:

Try `[parse]` and `[suggest]`

3

1 Use `[debug]` directive

Attach debugger to start debug session from terminal

```
ale.exe [debug] .\Zarlor.lnx --magnification 3 -f --controller keyboard
```

Possible error

Debug directive specified, but no process names are listed as allowed for debug. Add your process name to the 'DOTNET_COMMANDLINE_DEBUG_PROCESSES' environment variable. The value of the variable should be the name of the processes, separated by a semi-colon ';', for example 'DOTNET_COMMANDLINE_DEBUG_PROCESSES=ale'

```
"environmentVariables": {  
  "DOTNET_COMMANDLINE_DEBUG_PROCESSES": "ale"  
}
```

2 Pass arguments from `dotnet` to your CLI

```
dotnet run -- .\Zarlor.lnx --magnification 3 -f --controller keyboard
```

DragonFruit

Quick and easy way to create a simple CLI

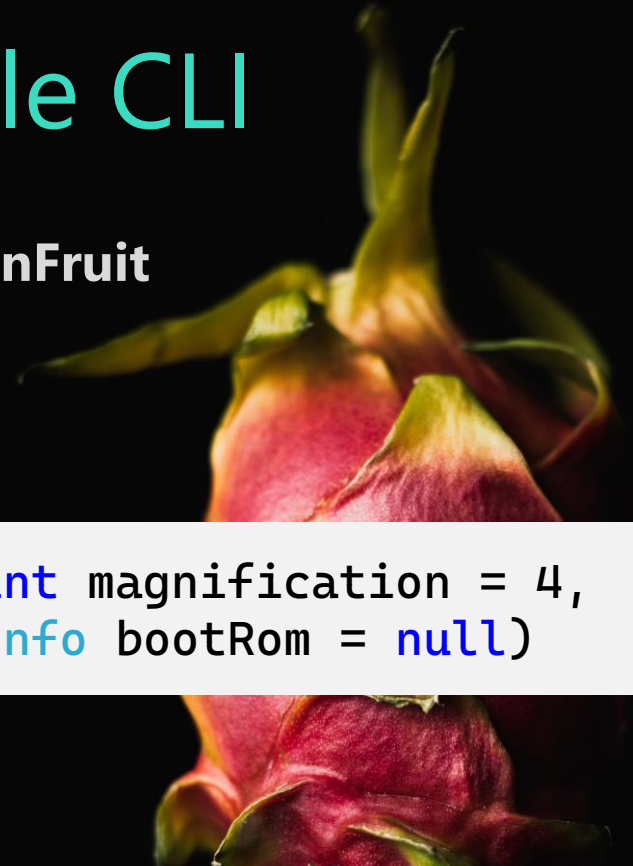
1 Add reference to NuGet package **System.CommandLine.DragonFruit**

2 Change Main method to include typed arguments
Must return `void`, `int`, `Task` or `Task<int>`

```
static int Main(FileInfo gameRom, bool fullScreen = false, int magnification = 4,  
    ControllerType controller = ControllerType.Keyboard, FileInfo bootRom = null)
```

3 Provide XML documentation as help information

```
/// <summary>  
/// Atari Lynx Emulator from DragonFruit  
/// </summary>  
/// <param name="gameRom">Game ROM filename</param>  
/// <param name="fullScreen">Run emulator fullscreen</param>
```



Demo

DragonFruit CLIs Debugging Using directives

Hosting integration

Extent hosted service with CLI

Configuration values

Service startup

```
dotnet run -- migrate --loglevel info --password $pwd -- --ENVIRONMENT Isolated
```

UseHost extension method

Access invocation context from `HostBuilder`

Adds `IHost` to `BindingContext`

Adds `InvocationContext`, `BindingContext`, `ParseResult` and `IConsole` to host Services

Propagate unparsed tokens host configuration and host builder factory

Flow [config] directives to host configuration

Bind parsed arguments to options

Output and rendering

Console is not your average console

Error, Out and In should be used

Avoid System.Console

Use provided abstraction instead (`IConsole`)

`TestConsole` in unit tests

Respect redirection

Input might come from response file

Output might be sent to file or elsewhere

Test output with pipes



Convenient:

Include `System.CommandLine.IO` for extension methods on Error and Out

Terminal GUIs for CL

Rich interaction with user

```
~/spectre.console .NET 5.0 main
dotnet run

Welcome

Spectre.Console is a .NET Standard 2.0 library that
makes it easier to create beautiful console
applications.
```

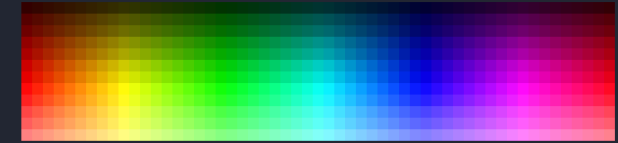


Remember:
CLI != GUI

Spectre.Console Features

Colors

- ✓ 2-bit color
- ✓ 3-bit color
- ✓ 4-bit color
- ✓ 8-bit color
- ✓ Truecolor (16.7 million)
- ✓ Automatic color conversion



OS Windows macOS Linux

Styles All ansi styles: bold, dim, italic, underline, strikethrough, reverse, and even blink.

Text Word wrap text. Justify left, center or right.

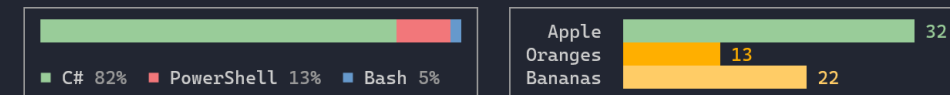
Lorem ipsum dolor sit amet, consectetur adipiscing elit. Quisque in metus sed sapien ultricies pretium a at justo. Maecenas luctus velit et auctor maximus.

Markup Spectre.Console supports a simple bbcode like markup for color, style, and emoji! 🍌 🍌 🍌

Tables and Trees

Foo	Bar
Baz	<div>Overview</div> <div>Files<ul style="list-style-type: none">src<ul style="list-style-type: none">foo<ul style="list-style-type: none">bar.csbaz<ul style="list-style-type: none">qux<ul style="list-style-type: none">corgi.txtwaldo.xml<div>3 Files, 225 KiB</div></div>
Qux	Corgi

Charts



Exceptions System.InvalidOperationException: Something went very wrong!
at Demo.ExceptionGenerator.SomeOperationGoingWrong() in
C:\Users\Patrik\Source\github\patriksvensson\spectre.console\examples\Console\Demo\ExceptionGenerator.cs:28
at Demo.ExceptionGenerator.SomeOperation() in
C:\Users\Patrik\Source\github\patriksvensson\spectre.console\examples\Console\Demo\ExceptionGenerator.cs:23
at Demo.ExceptionGenerator.GenerateException() in
C:\Users\Patrik\Source\github\patriksvensson\spectre.console\examples\Console\Demo\ExceptionGenerator.cs:12

+ Much more! Tables, Grids, Trees, Progress bars, Status, Bar charts, Calendars, Figlet, Images, Text prompts, List boxes, Separators, Pretty exceptions, Canvas, CLI parsing

Package CLIs as a tool

Important project properties

```
<ToolCommandName>al</ToolCommandName>  
<PackAsTool>true</PackAsTool>  
<Id>atarilynxcli</Id>  
<PackageOutputPath>./nuget</PackageOutputPath>
```



Global tools

Useable everywhere

```
dotnet tool install --global atarilynx --version 0.1.0
```

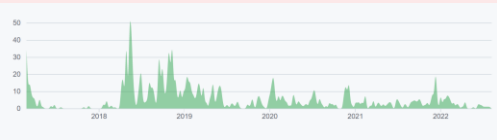



Local tools

Used from local context, such as CI/CD pipeline

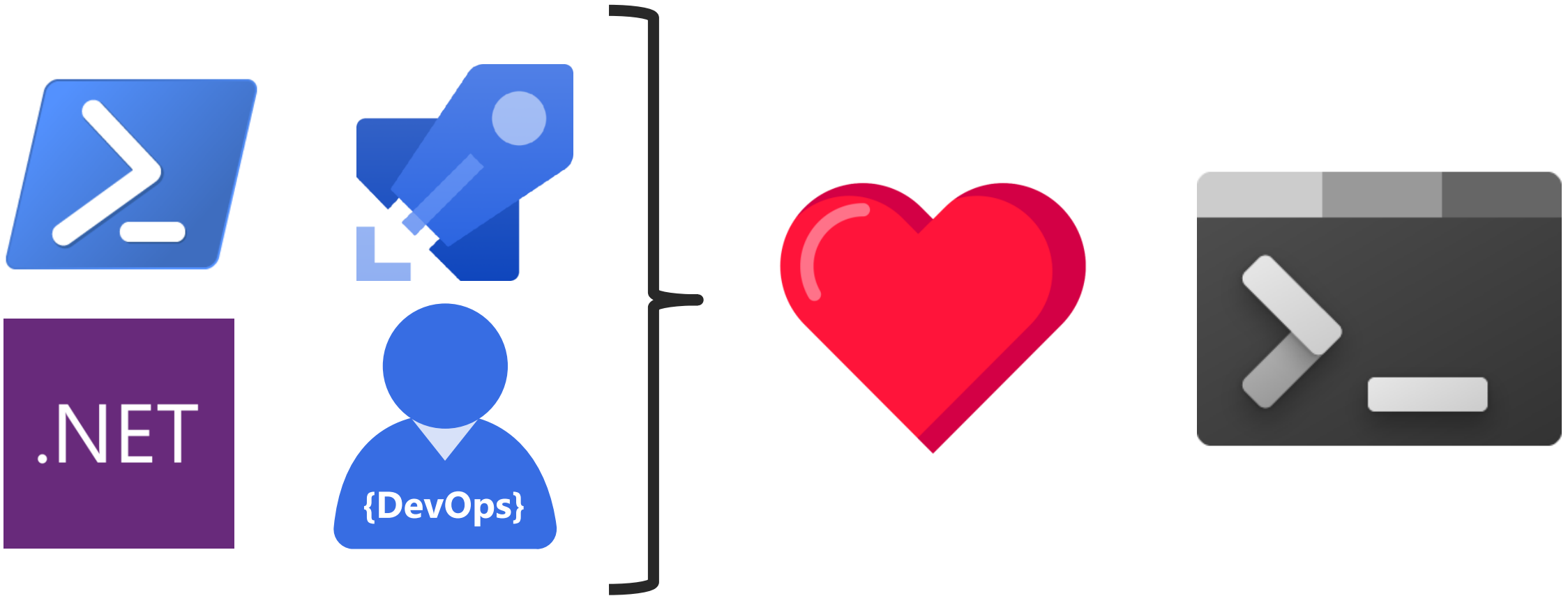
Requires tool manifest file

```
dotnet tool install atarilynx --version 0.1.0
```

Comparing CLI frameworks (opinionated)

	System.CommandLine	Spectre.Console.Cli	CommandLineParser	CliFx
Hierarchical commands	Yes	Yes	Limited	Yes
Suggestions and typo corrections	Yes	No	No	No
Help	Autogenerated	Autogenerated	Autogenerated	Autogenerated
Target framework	.NET Standard 2.0	.NET Standard 2.0 .NET 5.0	.NET Standard 2.0 .NET 4.0, 4.5, 4.6.1	.NET Standard 2.0, 2.1
Version	2.0.0-beta4.22272.1	0.46.0	2.9.1	2.3.1
Maintainer activity				
Binding	++	+	+	++
Host integration	++	+	-	++
Directives	debug, parse, config	None	None	debug, parse

Summary



\$ CLI <3 .NET --always

Resources

Command-line NuGet packages

<https://github.com/dotnet/command-line-api>

<https://github.com/Tyrrrz/CliFx>

<https://spectreconsole.net/>

Development

<https://www.microsoft.com/en-us/p/windows-terminal>

<https://dotnet.microsoft.com/>

<https://dotnetfoundation.org/projects>

<https://visualstudio.microsoft.com/>

Demo source code

<https://github.com/alexthissen/modernclis>

<https://github.com/alexthissen/atarilynxemulator>