Getting started

> **CAUTION**
> This part of the documentation exists for legacy purposes. All 3.x.x versions and above are not compatible with the instructions on this section.

Outline

Welcome to the docs, here you will find all the information you need to use this library.

You will find the following topics about the library:

- Homescreen visuals
- Write on the console
- Specific methods

And finally, you will find the precise documentation in "References" section.

> **NOTE**
> Feel free to contribute to the project by forking it and making a pull request or open an issue if you encounter a bug.

Structure

The library is composed of 4 main classes:

```
ConsoleAppVisuals
├──models
│   ├──Position.cs
│   ├──Placement.cs
│   └──FontYamlFile.cs
├──Core.cs
├──Extensions.cs
├──Matrix.cs
├──Table.cs
├──TextStyler.cs
└──Usings.cs
```

Usings.cs

This file contains the different usings of the library. It is used to import the different classes of the library and enable them globally in the library.

Core.cs

This class is the core of the library. It contains the methods to display the different visuals and variables.

Extensions.cs

This class contains different extensions methods for strings and tuples.

With Position.cs and Placement.cs, it belongs to the tools classes.

TextStyler.cs

This class is used to style the text. It contains the methods to apply a specific style to a text. Often used for the title. It may be useful to create your own style.

Table.cs

This class is used to create a table. It may be useful to display data in a table on the screen.

Matrix.cs

This class is used to create a matrix. It may be useful to display data in a matrix on the screen.

Position.cs

This class is used to define any position defined by an X and Y coordinate. It may be used in cases like matrix selectors for example.

Placement.cs

This class is used to define the placement of a text in the console. It may be useful to indicate where to place a text in a console, or to define the position of a text in a larger string.

FontYamlFile.cs

This class is used to define a font from a yaml file. It may be useful to create your own font.

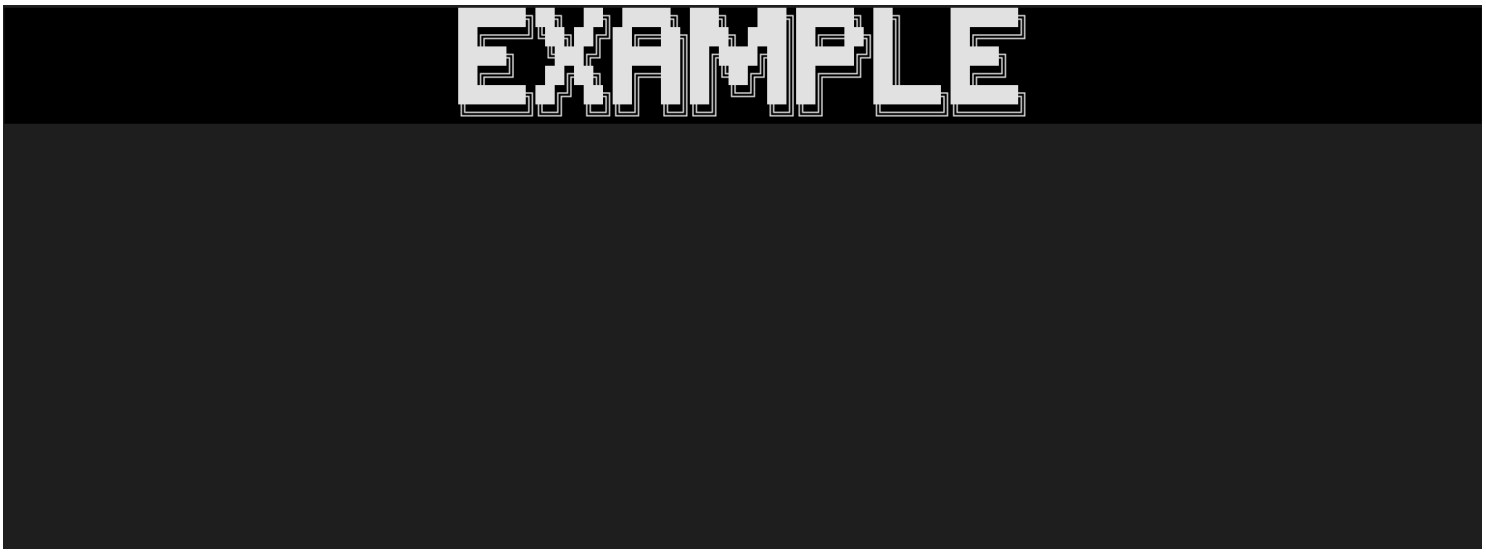
General displays for the home screen

Display a title

By default, no title will be displayed as no title has been set. You can set a title with the `SetTitle` method and then display it with the `WriteTitle` method.

```
Core.SetTitle("Example", 2);
Core.WriteTitle();

Console.ReadKey(); //[optional]: just to keep the console clean
```



Demo with an Example

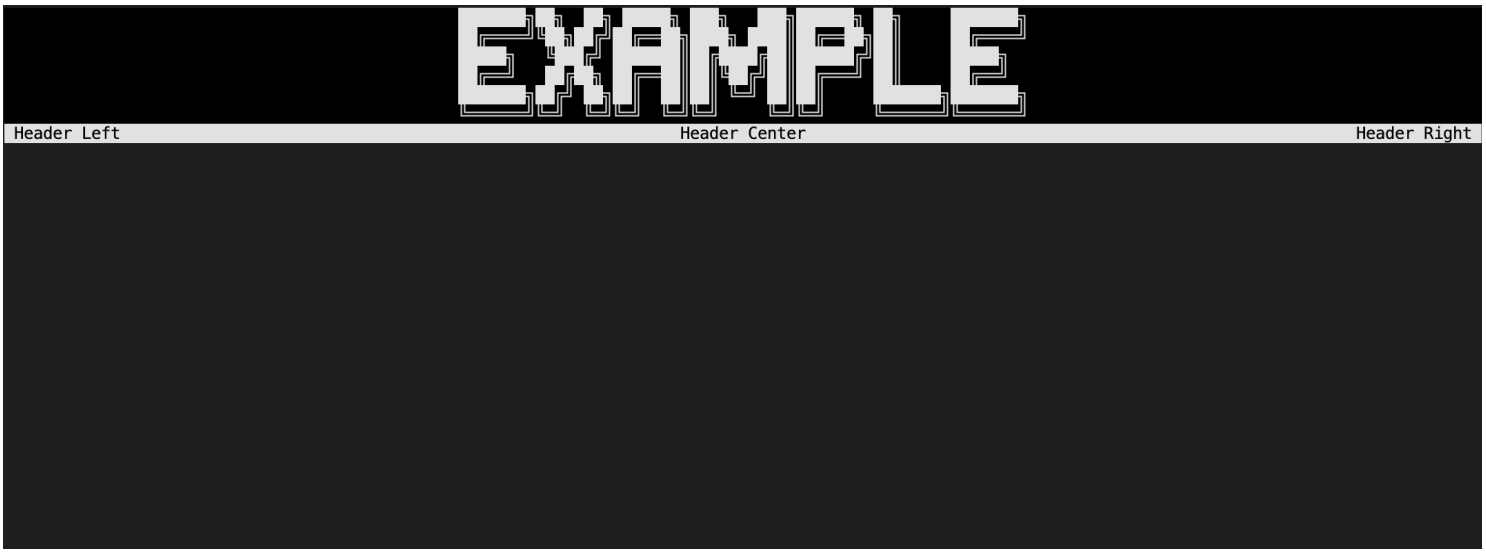
Display a banner

Now that we have seen the title, let's see how to display a banner. You may use the default arguments or define your own if you prefer an instant result, specify if you want to display the header or the footer or display your own banner.

```
Core.SetTitle("Example", 2);
Core.WriteTitle();

Core.WriteHeader();

Console.ReadKey(); //[optional]: just to keep the console clean
```

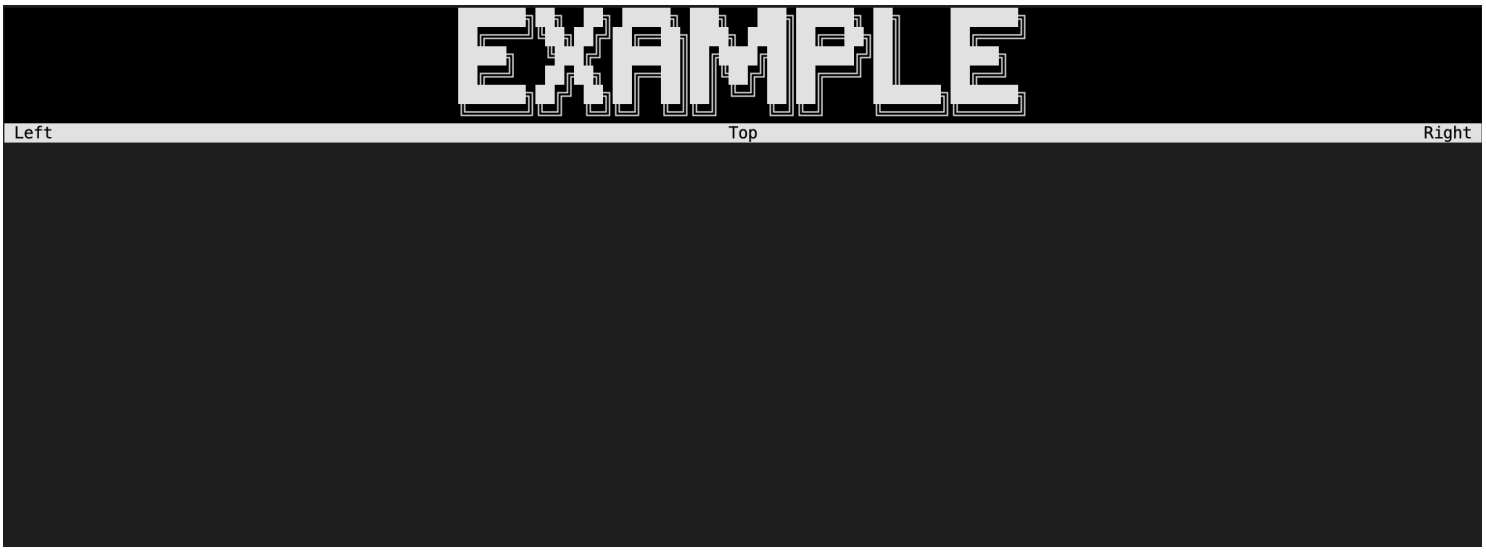


Demo with default arguments for the header

To customize the banner, you can change the arguments or change the default header and footer with the `SetDefaultHeader` or `SetDefaultFooter` methods.

```
Core.SetDefaultHeader(("Left", "Top", "Right"));
Core.WriteHeader(true);
Core.WriteFooter(true, ("Left", "Top", "Right"));

Console.ReadKey();
```



Demo with custom arguments for the header

Easy display

The `WriteFullScreen` method is the easiest way to display a banner and a title. It will display the banner and the title with the default arguments. Here is an example of what this method replaces:

```
Core.WriteFullScreen("Example");

// Instead of:
// Core.SetTitle("Example", 2);
// Core.WriteTitle();
//
// Core.SetDefaultHeader(("Left", "Top", "Right"));
// Core.SetDefaultFooter(("Left", "Top", "Right"));
// Core.WriteHeader();
// Core.WriteFooter();
//
// Core.ClearContent();
```

Writing on the console

Write a text in the console using placement

The `WritePositionedString` method is the most basic method of the library. It allows you to write a string in the console, with the possibility to specify the placement of the string within the width of the console.

```
Core.WriteFullScreen("Example");

Core.WritePositionedString("On the left", Placement.Left, default, 9, default);
Core.WritePositionedString("Centered", Placement.Center, default, 10, default);
Core.WritePositionedString("On the right", Placement.Right, default, 11, default);

Console.ReadKey();
```

Demo with placed strings

Including continuous printing

In addition to the placement, you can also specify if you want to print the string continuously or not. If you do, the string will be printed character by character, with a delay between each character. You may also interrupt the printing by pressing any key.

```
Core.WriteFullScreen("Example");

Core.WriteContinuousString("Hello World! Welcome to this beautiful app.", 10);

Console.ReadKey();
```

Demo with continuous printing

Including Multiple Lines

If you want to write a text with multiple lines, you can use the `WriteParagraph` method like this:

```
Core.WriteFullScreen("Example");

Core.WriteParagraph(default, default, "C# is a general-purpose, multi-paradigm programming language encompassing strong typing,","lexically scoped, imperative, declarative, functional, generic, object-oriented (class-based),"," and component-oriented programming disciplines.", "", "Press [Enter] to continue...");

Console.ReadKey();
```

Demo with paragraph

Include Styled Text

You can also use the `WritePositionedStyledText` method to write a styled text in the console. You may specify the placement, the color, the background color and the font of the text. The `StyleText` method allows you to style a string according to the font selected.

```
Core.WriteFullScreen("Example");

Core.WritePositionedStyledText(Core.StyleText("Hello World!"));

Console.ReadKey();
```

Including color

You can also specify the color of the elements and choose to apply the negative color to the text. Here are two example :

```
Core.WriteFullScreen("Example");

Core.ChangeForeground(ConsoleColor.Green);
Core.WritePositionedString("Hello World! Welcome to this beautiful app.",
Placement.Center, false, 10);

Core.ApplyNegative(true);
Core.WritePositionedString("Press any key to exit.", Placement.Center, true, 12);
Core.ApplyNegative(false);

Console.ReadKey();
```

Demo with color

> **NOTE**
>
> You may use the `ChangeBackground` method to change the background color as well.

Clear lines

Based on a line index and a number, you can clean several lines of your console. This is useful if you want to clean a specific part of your console. This way, you can choose to clean only the lines you want, and not the entire console.

```
Core.ClearLine(10); // Clears the line 10
Core.ClearMultipleLines(10, 2); // Clears the lines 10 and 11
Core.ClearContent(); // Clears the space between the two banners, header and footer
Core.ClearWindow(); // Clears the whole window with a continuous effect
```

Some properties

Heights

You may access to some useful properties concerning the heights of the visuals as:

```
Core.TitleHeight; // The height in the console of the title
Core.HeaderHeight; // The height of the header
Core.FooterHeight; // The height of the footer
Core.ContentHeight; // The height of the content
```

Catch updates

You will be able to catch if the screen has been resized with the `UpdateScreen` property. It will return a boolean indicating if the screen has been resized or not, or if the colors of the console has been updated.

Then you may use the UpdateScreen method to update the screen to the new values and reload the page.

```
if (IsScreenUpdated)
{
    UpdateScreen();
}
```

Selection cursor

You can also change the cursor character for every menu with the `SetCursor` method by precising the onward and backward characters.

```
Core.SetCursor('>', '<');
```

Own font

You may create your own font by creating a font file following the format specified in the source code (it includes, the three .txt files and the .yaml file). Then, you can use the `SetFont` method to globally set the font of your project.

```
Core.SetFont("/path/to/your/font/folder/");
```

> **WARNING**
>
> By default, the font is only used for the title. If you want other text to use the font, you have to do it manually using the `WritePositionedStyledText` method (for an array) or a simple `Console.WriteLine` is enough for a styled string.

Color panel

Finally, you may use the `GetColorPanel` property to get the color panel of the console.

```
var foregroundColor = Core.GetColorPanel.Item1;
```

Specific methods

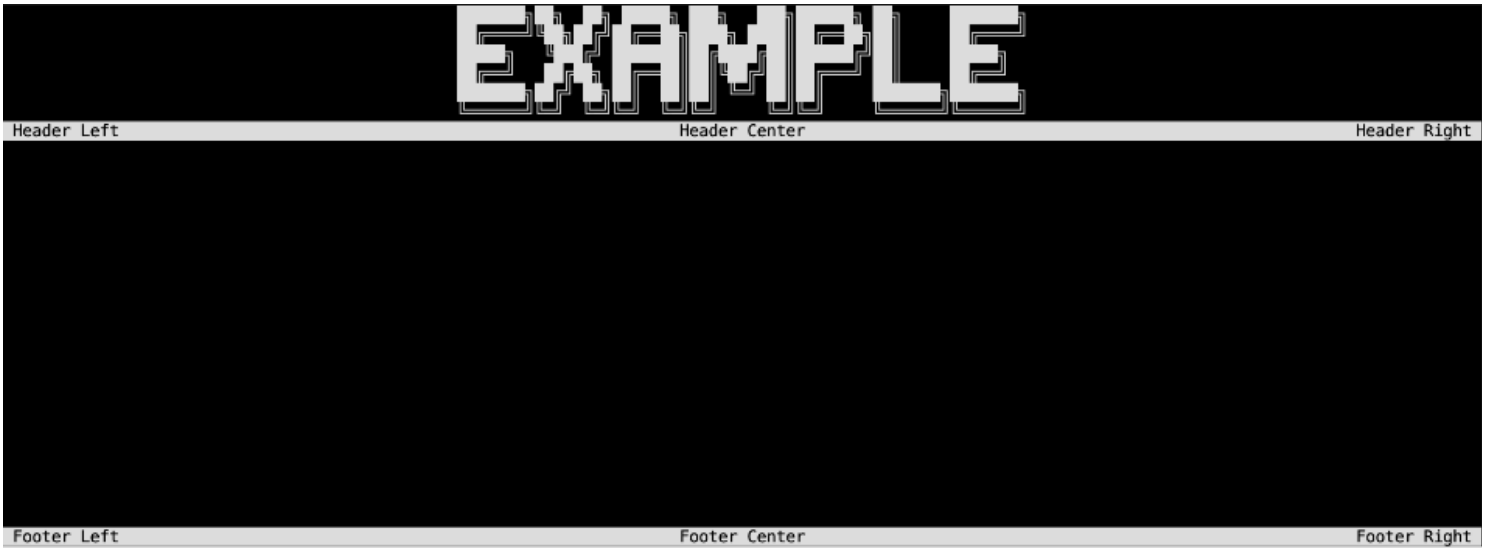
Scrolling menu

The `ScrollingMenuSelector` is a special block that allows you to display a menu with a scrolling effect. You may specify the question and the different choices.

```
Core.WriteFullScreen("Example", true);

Core.ScrollingMenuSelector("New question asked ?", 0, Placement.Center, null,
"Option 1", "Option 2", "Option 3");

Console.ReadKey();
```



Demo with scrolling menu

> **NOTE**
>
> To get the selected option and the key input, refer to the example project.

Number selector

The `ScrollingNumberSelector` is a special block that allows you to display a scrolling element with a number. You may define the minimum and maximum values, the step and the initial value.

```
Core.WriteFullScreen("Example", true);

Core.ScrollingNumberSelector("Please choose a number", 10, 50, 25, 5);
```

```
Console.ReadKey();
```

Demo with number selector

> **NOTE**
>
> To get the selected option and the key input, refer to the example project.

Prompt

The `WritePrompt` let you ask a prompt to the user and get the input. You may define the question and the default value.

```
Core.WriteFullScreen("Example", true);

Core.WritePrompt("Is your name John Doe ?", "John Doe");

Console.ReadKey();
```

Demo with prompt

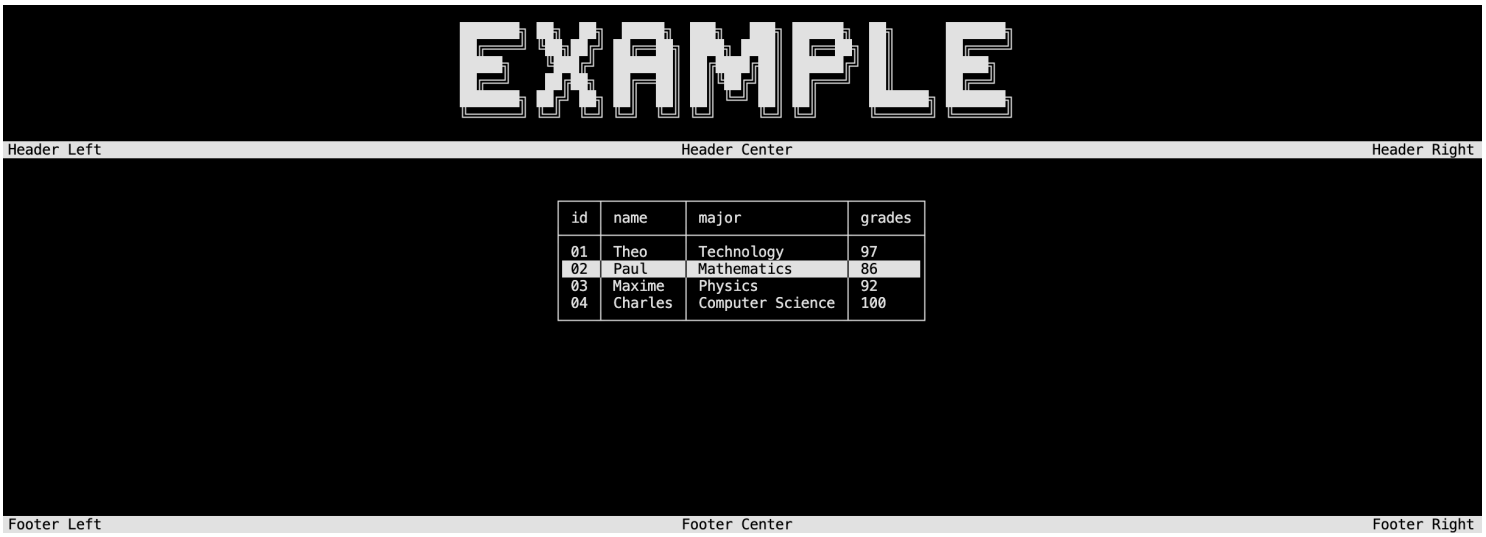
Table selector

First, you need to create a `Table` object giving the lines and optionally the headers just as in the example below.

```
List<string> headers = new () {"id", "name", "major", "grades"};
List<string> student1 = new () {"01", "Theo", "Technology", "97"};
List<string> student2 = new () {"02", "Paul", "Mathematics", "86"};
List<string> student3 = new () {"03", "Maxime", "Physics", "92"};
List<string> student4 = new () {"04", "Charles", "Computer Science", "100"};
Table<string> students = new (headers, new () {student1, student2,
student3, student4});
```

The `ScrollingTableSelector` is a special block that allows you to display the table with a selector.

```
students.ScrollingTableSelector(true, false, "Add student");
```



Demo with table selector

> **NOTE**
>
> Once you created the table, you can add, remove or update the data using the methods provided by the `Table` class (`AddLine`, `RemoveLine`, `UpdateLine`).

Here is an example of a table of how to use them:

```
students.AddLine(new () {"05", "John", "Biology", "95"});
students.RemoveLine(4);
students.UpdateLine(3, new () {"04", "Charles", "Computer Science", "55"});

students.Count
```

You may also use the `SetRoundedCorners` method to set the rounded corners to true or false for the tables.

```
students.SetRoundedCorners(true);
```

Matrix display

First, you need to create a `Matrix` object giving the data just as in the example below.

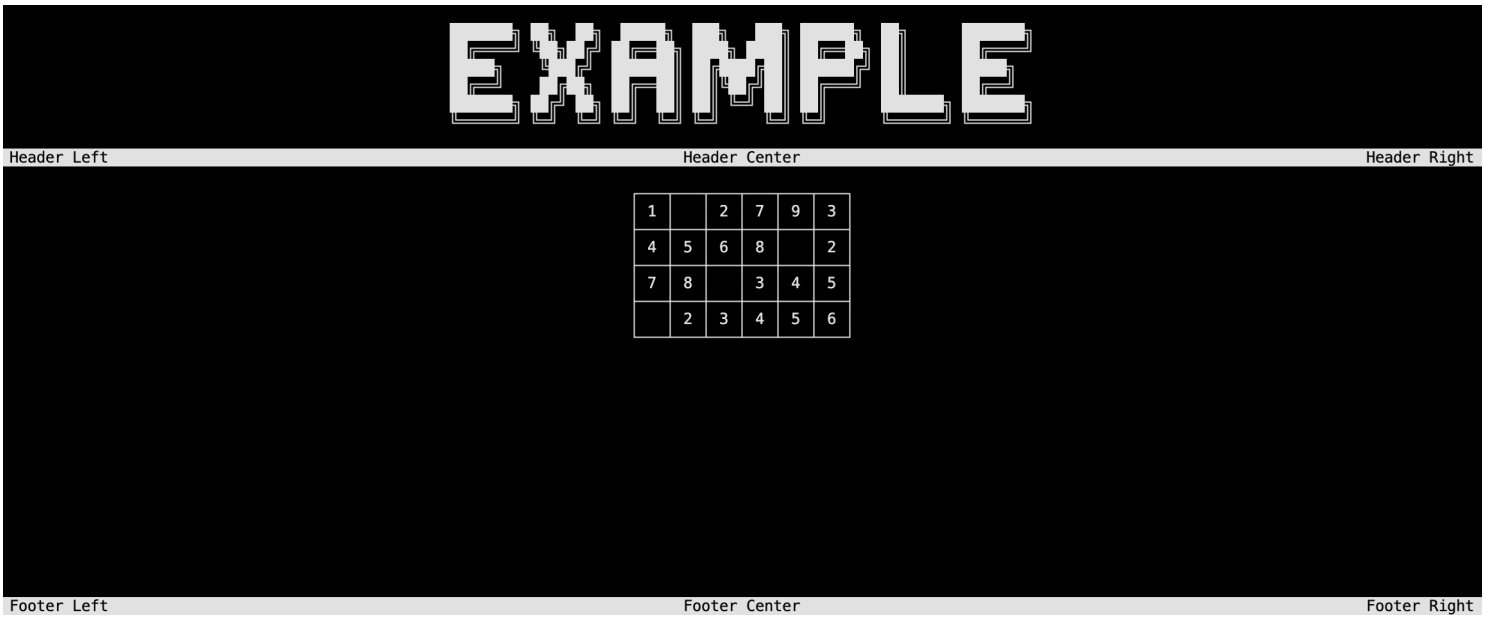
```
List<int?> firstRow = new() { 1, null, 2, 7, 9, 3 };
List<int?> secondRow = new() { 4, 5, 6, 8, null, 2 };
List<int?> thirdRow = new() { 7, 8, null, 3, 4, 5 };
List<int?> fourthRow = new() { null, 2, 3, 4, 5, 6 };
```

```
List<List<int?>> data = new() { firstRow, secondRow, thirdRow, fourthRow };
Matrix<int?> matrix = new(data);
```

The `WriteMatrix` is a special block that allows you to display the matrix. This is only visual, you can't select any element.

```
matrix.WriteMatrix(Placement.Center);

Console.ReadKey();
```



> **NOTE**
>
> Once you created the matrix, you can add, remove or update the lines using the methods provided by the `Matrix` class (`AddLine`, `RemoveLine`, `UpdateLine`) but also the elements using the `RemoveElement` and `UpdateElement` methods.

Here is an example of a matrix of how to use them:

```
matrix.AddLine(new () {2, 5, 7, 9, 3, 6});
matrix.RemoveLine(3);
matrix.UpdateLine(2, new () {3, 6, 8, 9, null, 2});
matrix.RemoveElement(new Position(2, 2));
matrix.UpdateElement(new Position(3,1), 7);
matrix.GetElement(new Position(2,1));

matrix.Count
```

You may also use the `SetRoundedCorners` method to set the rounded corners to true or false for the matrix.

```
matrix.SetRoundedCorners(true);
```

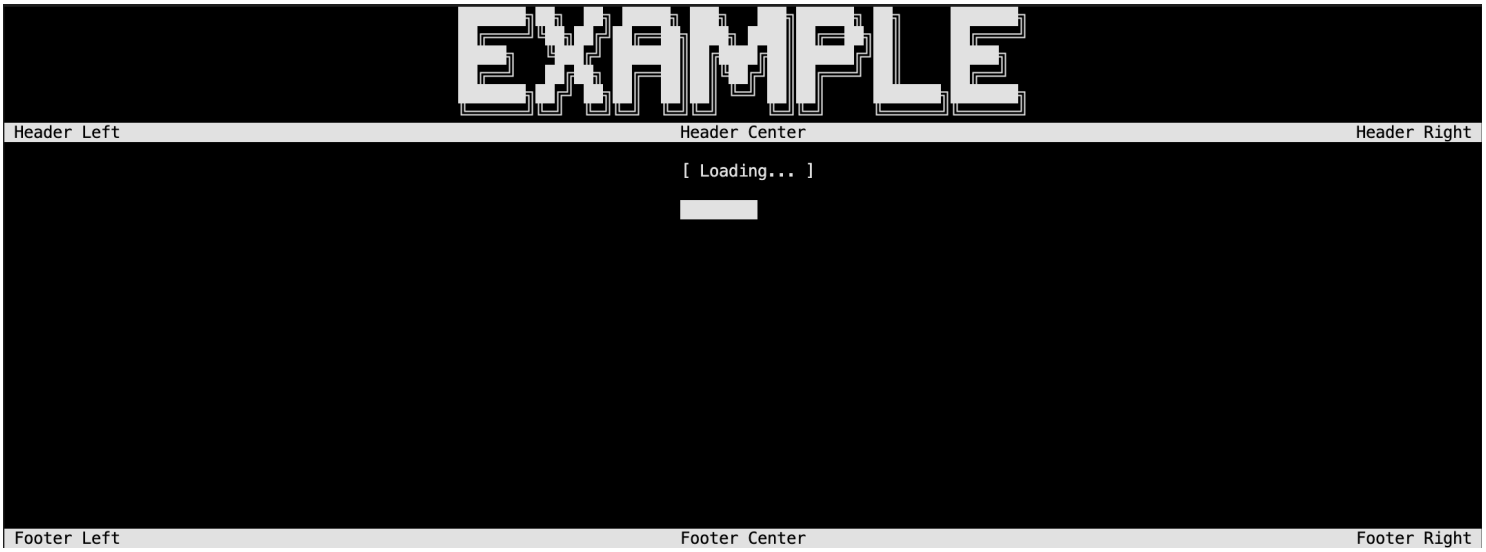
Loading bar

The `LoadingBar` is a special block that allows you to display a loading bar. You may define the text to display while loading.

```
Core.WriteFullScreen("Example", true);

Core.LoadingBar();

Console.ReadKey();
```



Demo with loading bar

Lawful loading bar

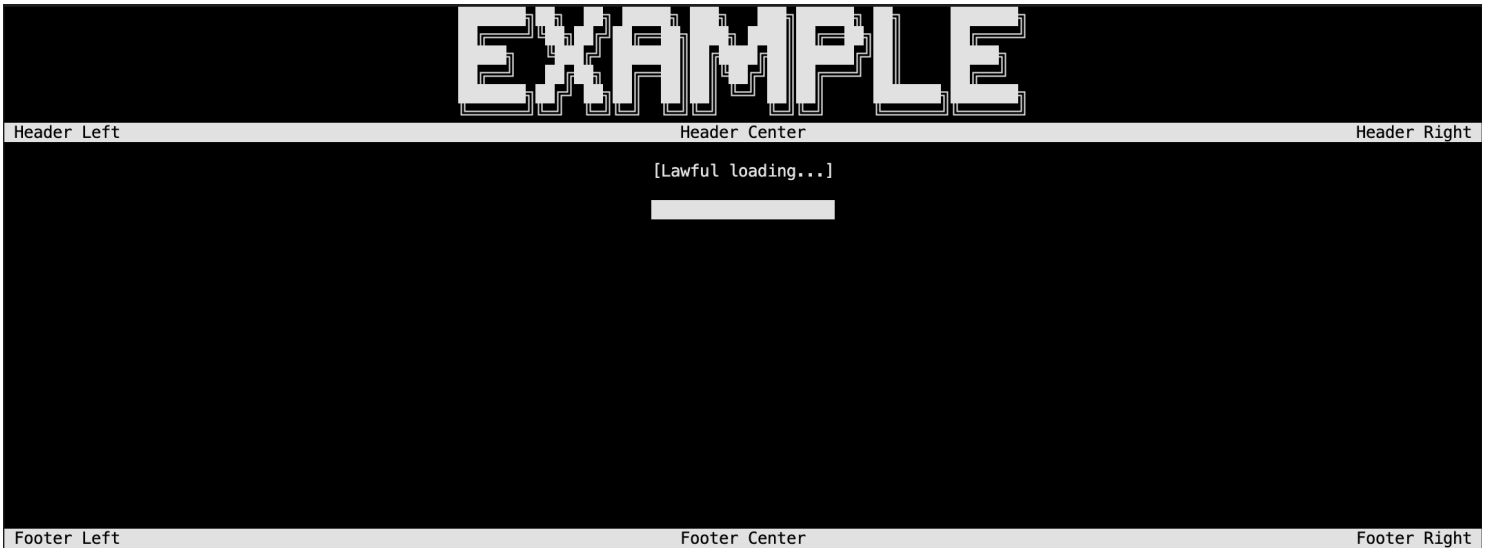
The `ProcessLoadingBar` is a special block that allows you to display a loading bar with a text and a *true* loading bar. You may define the text to display while loading.

```
Core.WriteFullScreen("Example", true);

var percentage = 0f;
var t_Loading = new Thread(() => Core.ProcessLoadingBar("[Lawful loading...]",ref
percentage)); // Create a Thread to run the loading bar on the console
t_Loading.Start();
while (percentage <= 1f)
```

```
{
    Thread.Sleep(100);
    percentage += 0.1f; // Simulate a loading process
}
t_Loading.Join(); // Wait for the Thread to finish

Console.ReadKey();
```



Demo with lawful loading bar

Exit

Last but no least, to exit the application, you can use the `ExitProgram` method. It will display a message and exit the application.

```
Core.WriteFullScreen("Example", true);

Core.ExitProgram();

Console.ReadKey();
```

Demo with exit message