RuntimeConsole v1.3

A. How to use the console at run time?
    1. Drag the Console prefab from the RTC/Prefabs folder into a canvas.
    2. Move the .dll-s from RuntimeConsole/Assets/Plugins folder into your Plugins folder
    3. Run the game and type RTC.Commands into the input field then press the submit button or just press enter on the keyboard then expand the log by pressing on the four arrows on the newly created log to see the available commands.
    4. The icons above the input field are for filtering the logs by their type.
    5. The X button above the input field is for clearing the filters.
    6. The stack button near the X button is for stacking/unstacking similar logs
    7. The trash button above the submit button deletes all the logs and clears the console.
B. How to use it in code ?
    1. To log something to the console window, you can use RTConsole.Instance.Logger.Add(new Log(logType, title, message, useStackTrace)); or much easier, use the RTC.Core.Extensions namespace and log like this RTConsole.Instance.Log(message); or check the other functions like LogWarning, LogError, LogException or LogFormat which works like string.Format
    2. To execute a command from code, type RTConsole.Instance.Execute(command);
    3. You can get information about a command by using RTConsole.Instance.GetCommand(command);

C. How to customize it ?
    1. Customizing the UI aspect and behaviour
        a. Select the "RuntimeConsole" game object which contains a ConsoleManager script that:
            - Can set the Toggle Key (open or close)

- Modify the Title Color and the log Icon for whatever log type you want by expanding the log types array. Here you can also add custom log types.

b. The "LogsContainer" game object
   - Can set the background color and the alternate background color for logs
   - Can set the Max Logs Count which will appear in the console window before they get recycled.

c. The "TaskBar" game object
   - Here you can add custom behaviour like log filters

d. The "InputBar" game object
   - Here you can change the Submit Key for executing commands
   - Can set the Command History Count (the maximum number of executed commands to remember before they get recycled) which can be navigated with arrows

e. The AutocompleteContainer game object
   - Here you can set the Max Suggestions Count (the maximum number of suggestions to appear when typing on the input field)

2. Extending the console's behaviour

a. Adding new log types
   - Create a static string which you will use when logging like this: Console.Instance.Log(staticString, message)
   - To assign a different icon and title color to it, go to the ConsoleManager game object and add a new log type which holds the name of the created static string.
   - If you want to add a filter button for this new log type, you must create a new UI image or button and add the LogTypeFilter component to it and of course, set the Log Type variable to the type you want to filter by, like you did in the ConsoleManager

b. Setting custom flags
    - To handle exceptions generated by commands set the RTConsole.Instance.HandleExceptions to true
    - To disable built-in commands set the RTConsole.Instance.EnableBuiltInCommands to false
    - To disable internal logs set the RTConsole.Instance.EnableInternalLogs to false

c. Creating commands (look at Demo/Scripts/DemoCommands.cs)
    - Create a static method and put the [Command] attribute above it
    - It is recommended that you set at least the Alias property of the attribute [Command(Alias = ‘‘name’’, Description = ‘‘description’’, Usage = ‘‘usage’’])
    - In order for this command to be recognized by the console, you must provide the console with the declaring type of the previously created command. (scroll down to ‘‘Additional Information‘‘ for the explanation)
    - To provide the console the newly created command, you must first create a class that implements the interface ITypesProvider which will return an array of typeof(declaring type) then you must set the RTConsole.Instance.Types to an instance of the newly created class.

d. Adding new handlers for custom parameter types (look at Demo/Scripts/DemoConverters.cs)
    - First you must create a class that implements the ITypeConverter interface, this contains two properties and a method:
        - Type : this returns the typeof(custom parameter type)
        - Regex : this is used to match the corresponding string for the Type
        - ConvertFrom receives the matched string corresponding to the Regex as the parameter. Here you must transform the provided string into an instance of an object of type Type.

- Secondly you need to add an instance of the previously created class to the RTConsole.Instance.TypeConverters list (it is recommended to add an instance of the generic ArrayConverter and ListConverter converters with that type as the parameter as shown in the DemoManager.Start method, if you plan to have lists or arrays of that type as a method parameter type)
- To replace an existing type converter, use the RTConsole.Instance.TypeConverters.Replace method or the extension RTConsole.Instance.ReplaceType.

Additional information:

- C.2.c explanation: Because scanning all types for commands at runtime can be really slow, you must provide the types that you know they contain custom commands, which will greatly improve performance.
- The built in logger can hold 100 logs by default, but you can use it's "Capacity" property to change that. This does not affect the displayed logs, which are stored in a circular buffer which recycles them.
- Don't add list converters using the default ListConverter if you want to run your game on WebGL . You can implement a custom ListConverter if you want it to work on WebGL.

Default syntax for the following types:

- Int, float, double, long, decimal and the unsigned version of them: a number (eg: 1.0)
- Bool: true/false or 1/0
- Char: a character enclosed in single quotes (eg: 'a')
- String: a string enclosed in double quotes (eg: "myString")
- Array: a series of a type from the above separated by space and enclosed in square brackets (eg: [1 2 3])
- List: same as array
- Vector3: { x y z }
- PlainText: plain text (convertible to string)

Suggestions:

- Create a static class where you place all your commands
- If you replace the syntax analyzer, you should replace the parser too
- Keep your console related code in separated classes, so when I change something, your code won't be replaced, and if the API will change, it will be easy to update your code.

Video tutorials:

- Creating commands: https://www.youtube.com/watch?v=rIf2Ajit4tM
- Registering custom types: https://www.youtube.com/watch?v=_A_us3Qr2YA
- Adding new log types: https://www.youtube.com/watch?v=89MzP_V6QSk

Website: https://emytofast.github.io/

Full code reference https://emytofast.github.io/Doc/index.html#!/article/home

Contact: miroiu.emanuel@yahoo.com