Configuration manager Specification

Configuration manager is a Pygame-UI configuration parser and handler used to configure components and their styles using XML configuration document. The configuration manager also manages the individual component styling using CSS-like properties.

Pygame-UI framework configuration handlers and classes

Vision

# Key Concept

The ConfigurationManager is a Pygame-UI utility that allows developers to manager the UI for their games/applications using XML markup. In order to make the UI visually attractive for the end user a lot of custom styling is required. ConfigurationManager allows for simple CSS-like styling for all components using an XML configuration file. The driving idea behind the ConfigurationManager is to reduce the need for programmatic initialization of UI components to the absolute minimum. ConfigurationManager would provide the possibility for creating a designer that would allow developers to use a visual tool to design their UI. This custom designer tool would provide the needed UI configurations.

Configuration Manager

# Architecture

The configuration manager is designed with the idea to have a single configuration file that drives the application UI configuration. This way a simple XML configuration can be provided to the application and the complex properties of the components would be initialized and handled using the configuration file. The manager takes the path to the XML configuration as a parameter in its constructor or initializes it from a default location. Then the configuration is loaded into memory and parsed on demand to reduce the overhead of parsing the data all at once. The developer needs to instantiate one single ConfigurationManager which would handle the initialization of styling managers and data managers for all components. Another important point is that the configuration manager can handle the parsing of all components as they are defined in the configuration and then instantiating them, which would be handled through the UIComponentCollection. This would be the feature that provides the developers with abstracted black boxing model for creating all of their UI. Currently the configuration file has to be written manually but once a designer is developed that would handle the generation of the XML configuration and then would generate a template for initializing the UIComponentCollection with a single line of code by the developer.

# ConfigurationManager Class

## Constructor

The ConfigurationManager class is instantiates by providing a path to the configuration file:

**ConfigurationManager(xmlPath='game.pyconfig') : return ConfigurationManager**

The constructor will raise an IOError if the file cannot be found or opened and a TypeError if the file cannot be parsed, meaning that the provided configuration is not a valid XML document.

## Methods

Once an instance of the ConfigurationManager class is successfully created the developer can take advantage of methods for instantiating other managers that are provided by the configuration manager and a method to get a list of all components that have been defined in the current configuration.

InitStylingManager([ID])

The InitStylingManager method takes as an argument the string ID of a component an instantiates a StylingManager for that component.

**ConfigurationManager.InitStylingManager(ID=componentId) : return StylingManager**

If the document element of the configuration has not been parsed yet and the structure of the document is not a pygame-ui compliant configuration, then a TypeError would be raised. If a styling node element is missing from the configuration, then the StylingManager cannot be instantiated and a TypeError is raised. If the component node is found in the configuration with the specified component id then a new instance of the StylingManager for that component is returned, otherwise None is returned.

InitStylingManagerByType([Node], [Type])

The InitStylingManagerByType method takes as arguments a component node from which the parsing should start and the type of the component that needs to be found and a StylingManager is instantiated for the first component in the configuration/in the current node tree that is found.

**ConfigurationManager.InitStylingManagerByType(node=None, type=None) : return StylingManager**

If no type is provided then an ArgumentError is raised. If no node is provided then the same steps as in InitStylingManager are taken except that the node is searched by type and not by id. If a node is provided the node’s child elements are parsed and if the component is found a new StylingManager is instantiated for it, otherwise None is returned.

FindAllComponents()

The FindAllComponents method parses the styling element of the provided configuration for all component nodes and returns a list of tuples containing component ids, component types and instantiated StylingManagers for the components.

**StylingManager.FindAllComponents() : return List[[string], [string], [StylingManager]]**

If the document element of the configuration has not been parsed yet and the structure of the document is not a pygame-ui compliant configuration, then a TypeError would be raised. If a styling node element is missing from the configuration, then the StylingManager cannot be instantiated and a TypeError is raised.

Styling Manager

# StylingManager Class

## Constructor

The StylingManager constructor takes as an argument a component node from the configuration document. If no node is provided a default instance of the StylingManager would be instantiated.

**ConfigurationManager.StylingManager(componentNode=None) : return StylingManager**

The StylingManager doesn’t parse all the starts at the time it is being instantiated. Instead it waits for a styling property to be requested and then instantiates all the styles. If no component node is provided to the constructor then the styling manager is fully instantiated by the constructor and default styling values can be changed using the styling properties.

## Methods

The StylingManager class provides utility methods with it.

InitStyles()

The InitStyles method forces the StylingManager to parse all the styles from the component node. It is called internally by the styling properties getters if the styles haven’t been instantiated already.

**ConfigurationManager.StylingManager.InitStyles() : return None**

If no component node has been provided to the StylingManager instance then this method would raise a TypeError.

ParseColor([color])

The ParseColor utility method is a parser that takes colors in different formats and returns a pygame.Color or an RGB color tuple. There is an internally defined color map which takes color string as input (e.g. “Aqua blue”), creates an RGB mapping if the color exists in the color map and returns a pygame.Color. The method also accepts hash-tagged RGB colors (e.g. #FF3030, #000) and returns a pygame.Color. If a color tuple is provided as input then the same tuple would be returned.

**ConfigurationManager.StylingManager.ParseColor([color]): return pygame.Color**

If a color string which the color map does not contain is provided, then a KeyError would be raised. If the argument provided is not a color string, tuple or hash-tagged RGB value then an ArgumentError would be raised.

## Properties

* Name: **dimensions**

Type: Integer tuple

Action: Gets/sets (width, height) dimensions in pixels

* Name: **width**

Type: Integer

Action: Gets/sets width in pixels

* Name: **height**

Type: Integer

Action: Gets/sets height in pixels

* Name: **position**

Type: Integer tuple

Action: Gets/sets (top, left) position in pixels

* Name: **top**

Type: Integer

Action: Gets/sets top offset in pixels

* Name: **left**

Type: Integer

Action: Gets/sets left offset in pixels

* Name: **background\_image**

Type: String

Action: Gets/sets a path to an image

* Name: **background\_color**

Type: String/RGB tuple

Action: Gets/sets a color value accepted by the color parser

* Name: **color**

Type: String/RGB tuple

Action: Gets/sets a color value accepted by the color parser

* Name: **font\_size**

Type: Integer

Action: Gets/sets a point size value for the font size

* Name: **font\_family**

Type: String

Action: Gets/sets the font family for the text

* Name: **text\_align**

Type: String

Action: Gets/sets the text align string

* Name: **vertical\_align**

Type: String

Action: Gets/sets the vertical align

* Name: **visibility**

Type: String

Action: Gets/sets the visibility

* Name: **xml\_node**

Type: XML Node

Action: Gets/sets the component XML configuration node

## References

<http://www.w3.org/TR/CSS21/>

<http://en.wikipedia.org/wiki/List_of_colors>