Dathomyhria 2014

7-SEGMENT DISPLAY UNIT

CYBERCONIAN | OCLANAY

CONTENTS

INTRO & WELCOME

Dear Valued Customer,

It is my pleasure to present and offer you a simple and potentially useful component that will add functional variety to your virtual reality.

During the course of the product's development, I had a personal need for a simple seven segment display system. Being a person of the older Generation-Y as is so-named, these old school display units still hold an intriguing appeal to me, particularly in virtual environments that are technology-centric.

As many of you know, seven segment displays have been around since the RMS Titanic set sail, but have not seen widespread deployment until the early 1970s. They are frequently seen in clocks, exchange rate boards, digital metres and other metric devices. They are sometimes seen in electronic keypads and hotel safes, as well as infamous digital watches.

This package includes a single seven segment display (7SD) prototype. As an essential bonus, the colons (:) are also included. You may find use for the colons indicator when creating clocks of various sorts but are not required to make the displays function. With the 7SD, you can duplicate it in groups, colourise and drive them with a string input that will suit common use cases in your world.

Most importantly: *you can use any textures you like. You don't even have to have seven segment display-like textures. You can have any 'digits' style you like that can easily bear little relevance to the old school displays! This opens up many stylish and practical uses for this display device, giving it a wonderful array of different yet valuable uses.*

My small informal organisation has produced a variety of 3D aircraft since VR became more apparent in the late 1990s on a limited scale. Today, I am happy to extend my accumulated VR experience to you. The product is designed specifically for Unity™ and is highly optimised and is suited for scaling in your scene.

With an optimised scaling effort built in, you can use hundreds of these units for complex display boards in your scene with minimal impact to computer performance.

Should you feel the need for a degree of helpful support, the company forum is available for your perusal.

Once again, I wish to express my gratitude and thanks, for letting me offer you this asset that endeavours to increase the real value of your great virtual worlds that are in your life.

I am glad to offer those worlds more.

[illegible]

Christian Kanakis
Technician @ **Cyberconian | Oclanay**
Sydney, Australia

INSTALLATION

Simply do an import of a custom package, selecting the **Oclanay_7SD.unitypackage** file. You will be prompted to select the items to import. Please select all the items, and the 7SD will be imported into your assets folder. This is all you have to do.

The folder is named ***_Oclanay_7SD*** *which will contain the entire product and this document therein. This package will not overwrite standard assets, nor change any fundamental configurations of the operating software, nor interfere with any other asset. The product is fully self-contained, and can survive moves and renames throughout the asset hierarchy of your project. The operating code is also self-driven, and can operate independently of your scene.*

UN-INSTALLATION

Simply delete the **_Oclanay_7SD** folder (or whatever name you have given it if it was renamed). After a confirmation prompt, the product will be removed from your project and related scenes. This is all you have to do.

Please be aware that any custom or third party components that rely on the products existence, whether driving the product or relying on relevant event handlers from the product, those components may encounter technical problems. It is advised to check any dependent components you might have that drive or are driven in some way by this product sequel to un-installation.

GENERAL DISPLAY USAGE

INSTANTIATION

Drag the prefab 7SD from the _Oclanay_7SD folder into the project hierarchy or a scene window. What you will see is a dark-red seven segment display. Notice in the project hierarchy, that there are a number of items under the 7SDG (7 Segment Display Group) which constitute the product. In particular, notice the “display” item. If you wish to have multiple displays under this group, please duplicate the display object. Do not move it anywhere else in the hierarchy, and please do not rename it. The reason for this is so that the input string, which is entered into the INPUT field, can render the characters correctly.

You can programmatically create new 7SDG prototypes as well using any supported language of your choice. A technical explanation of programmatic instantiation is beyond the scope of this manual.

OPTIONAL COLONS

Drag the prefab 7SDC from the _Oclanay_7SD folder into the project hierarchy or a scene window. The product comes with an optional display of two dots that people are familiarly with that is often seen in digital clocks. These are referred to as the 'Colons' display component as it is formally called. Like the seven segment display, the colons can be customised in terms of display and dimmed colours, dot flashing frequency (if so desired) and can be enabled or disabled. Each dot can be individually enabled and are controllable either through the Editor or by programmatic means.

CHARACTERISING THE DISPLAYS & COLONS

The displays can be characterised in a number of ways as follows:

- Displays are grouped under the "7SDG" Game-Object
- Each 7SDG group may consist of one or more displays.
- Each 7SDG group is fully self contained and does not interfere with other 7SDG objects
- Characters are a string which represents the input for all displays in a given 7SDG group
- Each character of the input represents a single display
- Every display that is duplicated will use the next input string character
- Characters are displayed from the right most toward the left most but can be reversed by checking an option in the editor
- Empty character strings or strings containing unrecognised characters are not displayed
- The decimal point can be used by specifying dots (full stops) as part of the input/data string for the display group
- Displays and Colons can be driven programmatically through the API
- Displays can update at a certain rate every second
    - Useful for automatic counters, clocks and timers
    - Great for real time reading of rapidly changing data
    - Rate of update can be configured on a so-many-time-per-second basis
    - Consumes more CPU and GPU time due to the update privilege
- Displays can also update when the input changes independently
    - Good for infrequent changes, such as exchange rate boards, minute based clocks or weather information
    - Consumes little CPU and GPU resources due to the infrequently animated state of affairs of the devices
- Face colour, dim digit and digit colours can be customised
- An event handler can be used to respond every time the display updates
    - Only valid with "Always Update" mode
- Device can be put into 'test' mode (including the Colons display)
    - A counter like effect is seen, showing the affected display group in count-up mode
    - The colons are solidly switched on, but do not animate

GROUPING AND DISPLAY INSTANCES

- If you want to have a number of displays grouped together, they would have a common 7SDG group parent. For instance, suppose you want to display n characters in a group for an input string, then you would have n displays. Each group has its own input string, whose characters apply across the number of displays available.
- If there are not enough displays or the text string is not long enough, then only the first portion of the string is displayed that matches the number of displays, or the unused displays are simply blank.
- If you have different groups of displays, they will function independently of each other. Each group will have its own input-string.
- You can duplicate each 7SD group in the editor and the new duplicate will operate independently of the other groups present in your scene.

API REFERENCE AND DRIVER DETAILS

DISPLAYS

TEST MODE

void TestMode (Boolean test)

When set to true, the device goes into self test covering the entire display group. The default value is false. When enabled, any other settings you have made to the device is overridden by the test program.

DATA

void Data (String characters)

This represents the Input Data. This should always be used programmatically, particularly if your displays will change during the lifetime of your virtual environment. Every time the Data property is called, the displays will update to the value of the input string.

You can use Input and Data together - Input to start the display and Data to update. Of course, you can use **Data** property to start the displays as well, which is the recommended method.

INPUT

void Input (String characters)

This sets up the displays with the input data. This is useful when you do not have a need for the displays to change during the virtual environment lifetime. This field can also be set in the editor. Note that once the environment has started running, further calls to input will have no effect on the output of the rendered digits on the displays.

BACK COLOR

void BackColor (Color32 color)

Sets the colour of the dimmed area of the display. By default this is dark red with even darker unlit segments. Changing this property whilst the scene is running will also update the colours.

DIGIT COLOR

void DigitColor (Color32 color)

Sets the colour of the segments of the display. By default this is bright red when the relevant segments are lit. Changing this property whilst the scene is running will also update the segment colours.

~X~

ID

void ID (String identifier)

Optional. The literal ID of the device group. It is an alternative to Game Tags, and may be useful in third party programs that could get specific collections of displays without needing an interfering Tag in a scene.

SERIAL

void Serial (Int serialNumber)

Optional. The numeric serial number of the device group. It is an alternative to Game Tags, and may be useful in third party programs that could get specific ordered-collections of displays without needing an interfering Tag in a scene. Such a use could be a series of displays that read sequential data such as a list of exchange rates, or high scores.

ONUPDATE EVENT

void EventHandler (Object sender, EventArgs e)

Responds to the device every so often if Always Update is enabled (or set to true). The **UpdateRate** determines how many times per second the **OnUpdate** event is called. It also determines how frequently the displays are updated. Please note that a higher value (especially in light of many active displays in your scene) will consume more resources and degrade performance.

UPDATERATE

void UpdateRate (float frequency)

The frequency in Hz, which the displays are rendered and updated. This is only useful when Always Update is checked otherwise this figure has no effect. Default value is 1 Hz (once per second, useful for common counters and digital clocks with the seconds display running).

SEGMENTS

void Segments[] (Texture2D[] textures)

This is an array that stores the textures. This is used primarily by the editor. This is where you assign static textures which all display devices share.

Please do not use this property programmatically, unless you wish to change the texture reference at run time. Each index represents a mappable character, pointing to the relevant texture representing the character.

REVERSE STRING

void ReverseString (Boolean isReversed)

Default is false. Setting this to true will cause the input string to be processed in reverse. This is used by the displays in terms of their rendering order. Typically this is right to left. Setting this to true will thus cause the text to be rendered to the devices left to right instead.

COLONS

ENABLED

void Enabled (Boolean isEnabled)
Sets the colons as to whether they are active (Enabled = true) or inactive (Enabled = false). If the device is disabled, no other properties will have any useful effect despite their states.

BACK COLOR

void BackColor (Color32 color)

Sets the colour of the dimmed area of the display. By default this is dark red with even darker unlit dots. Changing this property whilst the scene is running will **not** update the colours.

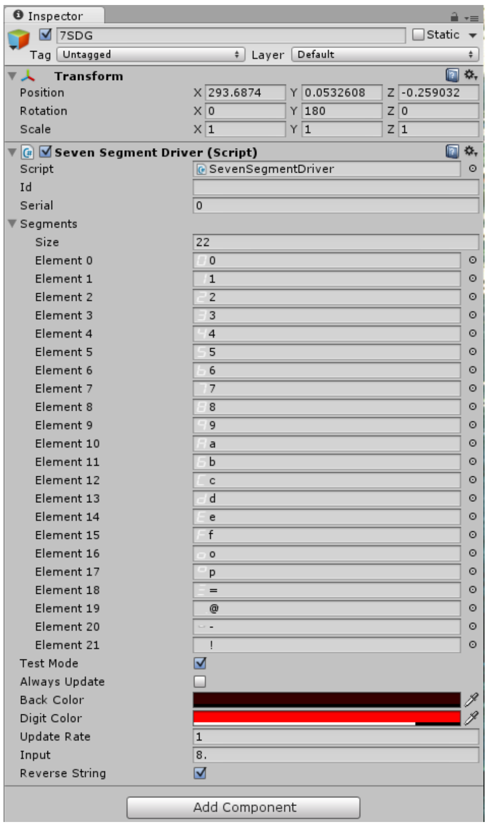
DIGIT COLOR

void DigitColor (Color32 color)

Sets the colour of the dots of the display. By default this is bright red when the relevant dots are lit. Changing this property whilst the scene is running will **not** update the segment colours.

IN-EDITOR PRODUCT REFERENCE

7 SEGMENT DISPLAY



```
7SDG                                   [ ] Static
Tag  Untagged                Layer  Default

Transform
Position        X 293.6874   Y 0.0532608   Z -0.259032
Rotation        X 0          Y 180         Z 0
Scale           X 1          Y 1           Z 1

[✓] Seven Segment Driver (Script)
Script          SevenSegmentDriver
Id
Serial          0
Segments
  Size          22
  Element 0     0
  Element 1     1
  Element 2     2
  Element 3     3
  Element 4     4
  Element 5     5
  Element 6     6
  Element 7     7
  Element 8     8
  Element 9     9
  Element 10    a
  Element 11    b
  Element 12    c
  Element 13    d
  Element 14    e
  Element 15    f
  Element 16    o
  Element 17    p
  Element 18    =
  Element 19    @
  Element 20    -
  Element 21    !
Test Mode       [✓]
Always Update   [ ]
Back Color
Digit Color
Update Rate     1
Input           8.
Reverse String  [✓]
```

The image above shows the Unit in its default mode when placed in the editor or instantiated programmatically.

You do not need to concern yourself with the Segments section. You should only use this when expanding the number of digits or changing the style of the textures entirely. To learn more about this, please see the section 'Extending the 7SD'.

SCRIPT

This should never be changed. It is always set to ***SevenSegmentDriver***. The C# file is found in the scripts directory of the product. If you update the script, this voids any support from Cyberconian | Oclanay.

SEGMENTS

The collection of textures used to render the digits. There are 22 textures assigned by default. Each character in the input string takes the first letter (or number) of the texture name and is picked for rendering on the relevant display unit.

Currently, the characters that are supported are: 0,1,2,3,4,5,6,7,8,9,a,b,c,d,e,f,o (degree symbol raised), p (degree symbol lowered), = (three horizontal segments enabled), @ (decimal point only), - (dash symbol, also used for negative numbers) and ! (the empty/null texture).

You can add more (or remove) textures if you wish. Remember that if you remove any textures, then any mapped characters will no longer be shown on the display. If you add more textures, the first letter of the unique texture name will be used.

The first letter cannot already be present in other texture names; it goes to show you will only have non numbers and most alphabetical and punctual characters for display, as the mapped characters are already taken by default.

TEST MODE

When this is ticked, a counter will appear on the display. If there are multiple displays in a group, then those displays will partake in the counting. It is a simple counter, counting from 0 to infinity. It is there to ensure the virtual display works like it would in the real world. Disable this when it is not needed the API will have no effect otherwise.

ALWAYS UPDATE

When enabled, the display will enter a counter-like mode. That means the display is called at a fixed-time interval. Typically, you would only use this when programmatically interacting with the display group. You would assign an event handler to the instantiated 7SD group class which would handle what happens to the display every time the display driver is called internally. It is recommended that you write code that simply calls the Data property for continued updates and Input for one-time setups. This method is provided as a convenience service to the displays, but you are encouraged to write your own updaters to suit your gaming environment.

INPUT

This sets the initial characters to be seen on the displays. You can have as many characters as you like, but please be aware that you will also need the matching number of displays. Suppose if you want to write: 123.45, you would need 5 displays. Note that the decimal point (.) would render on the third display.

Any additional characters are simply not shown on the displays if there are not enough displays in the scene.

Note that characters are displayed from right to left, so in the example of 123.45, 5 would be shown on the first display, then 4, then 3 (with the dot), then 2, then 1.

REVERSE STRING

This reverses the Input data. Your data string will not be altered, simply reversed. You do not need to do anything further. Note that the Input will, when Reverse string is checked, be rendered left to right instead of the default behaviour being right to left.

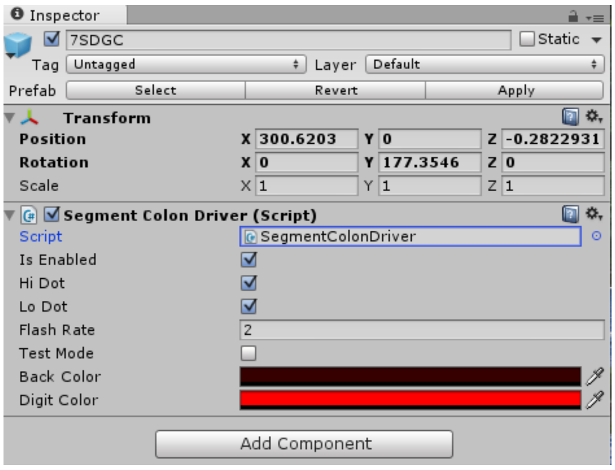
PERFORMANCE CONSIDERATIONS

The product is highly optimised within Unity(tm) Indie for scaling performance. There are acti ons you can take to improve optimisation further at the expense of product quality. The factory setting is a well optimised balance between quality, performance and potential display quantity up to around 1024 displays. Note that this is a general limit on a performance based machine, and it is not recommended that more than 512-768 displays be used in a scene at any one time.

Using in excess of 1024 individual displays will have a noticeable impact on frame rates. Displays that frequently update (such as more than once per second) will have an even greater impact. It is recommended that 512 displays be used in situations where displays are vigorously animated.

You can enhance performance by disabling the block component (the thing which makes the bulk of the display), and if you have Unity(tm) Pro, using Static batching (although you cannot use this on the 'display' object). You can reduce the quality of the textures as well including the filtering and aniso level in the texture settings. Textures are 1024 in height, but you are free to modify them for personal use that suits the nature of your virtual environment.

COLONS



Inspector
7SDGC ✓ Static
Tag Untagged Layer Default
Prefab Select Revert Apply

Transform

Position	X 300.6203	Y 0	Z -0.2822931
Rotation	X 0	Y 177.3546	Z 0
Scale	X 1	Y 1	Z 1

Segment Colon Driver (Script)

Script	SegmentColonDriver
Is Enabled	✓
Hi Dot	✓
Lo Dot	✓
Flash Rate	2
Test Mode	☐
Back Color	■
Digit Color	🟥

Add Component

SCRIPT

This should never be changed. It is always set to ***SegmentColonDriver***. The C# file is found in the scripts directory of the product. If you update the script, this voids any support from Cyberconian | Oclanay.

IS ENABLED

Determines whether the device is active or not. if it is not enabled, the colons will not show, and will appear to be in the 'off' state regardless of other parameters and their states.

HI DOT

Enables the upper dot. If it is disabled, the upper dot is never lit.

LO DOT

Enables the lower dot. If it is disabled, the upper dot is never lit.

FLASH RATE

Measured in Hz. This represents the number of times the colons flash on and off. By default, the setting is 2 Hz. This means the dots are on for half a second and off for half a second. The default was picked to be 2 since the most common application of the colons device is for digital clocks and stop watches. Change this value for faster rates by increasing it above 2, and for slower rates by decreasing it below 2. Setting this to 0 will keep the displays switched on permanently during the life time of your played scene.

TEST MODE

Shows both dots on. If this does not happen when your scene is played, something has gone wrong with the driver or you have not setup or instantiated the device correctly.

BACK COLOR

The background colour of the device. By default this is dark red with even darker dots.

DIGIT COLOR

The colour of the lit dots. By default they are bright red. This property has no effect when the dots are not switched on.

TECHNICAL SPECIFICATIONS

DISPLAY

- Physical Device Aspect Ratio (W/H): **1.635 (approx)**
- Common texture information: 300x420x32 alpha png @ 495kB each
- Textures are uncompressed true colour with Aniso level 4, trilinear filtered with clamping
- Maximum operating instances (50% fps loss): 1024 (512 animated)
- Maximum operating instances (90% fps loss): 2048 (1024 animated)
- Unity™ Indie compatible
- Maximum visual rate of change: 1/1000 second

COLONS

- Physical Device Aspect Ratio (W/H): **0.297 (approx)**
- Common texture information: 50x420x32 alpha png @ 4kB each
- Textures are uncompressed true colour with Aniso level 4, trilinear filtered with clamping
- Unity™ Indie compatible
- Maximum visual flash frequency: 60 Hz

ACKNOWLEDGEMENTS

PRODUCT FUNCTIONALITY (SEVEN SEGMENT DISPLAY WITH DRIVERS)

Backed by the **Cyberconian** Group, Sydney, Australia.

3D ENGINE SOFTWARE (UNITY INDIE)

Provided by **Unity Technologies**

ALPHAILLUMN SHADER (COMPRISING THE SEVEN SEGMENT DISPLAY MATERIALS)

http://wiki.unity3d.com/index.php?title=AlphaSelfIllum, Bill Vinton (Sync1B) **AlphaSelfIllum.shader**

SEGMENTED TEXTURES (SEVEN SEGMENT DISPLAYS AND DECIMAL POINT)

http://graphicriver.net/item/vectorized-seven-segment-display/3840269,

Author: DesertRain, Austria (Vectorised Seven Segment Display)

PLANE GENERATOR (SIMPLE DISPLAY-BACKPLANE OBJECT)

Author: http://wiki.unity3d.com/index.php?title=CreatePlane, Michael Garforth.

Extended by Juan Manuel Palacios and Frogsbo.

I.P COPYRIGHTS DECLARATION

SUPPORT

Limited Technical support is provided by Oclanay. Oclanay cannot provide custom advice on how to code, build or integrate the product with other systems, but can advise on the individual operation of the virtual devices on their own.

If you are looking for integration and general programming ideas, then community support can be freely obtained. The support forums are located at http://oclanay.cyberconian.com/support/forum. You are encouraged to use the free service to obtain support from Cyberconian and other patrons using the product.

VERSION HISTORY

1.0.0.0

1. Initial release of 7SD Product