Anaface: AnaGame’s 2D UI engine

Anaface is AnaGame’s User Inferface engine, holding classes that represent controls that draw themselves to the screen and respond to user input. In terms of the library stack, it is located in the middle, with two libraries below it and two libraries above it.

Anaface (in Windows) uses Direct2D and DirectWrite to render controls (and the Video Graphics library and Direct3D in the case of TArena controls).

Before you can build Anaface, you have to have the TrecLib and VideoGraphics libraries built.

# Set-up

Obviously, this is a rehash of what might be found in a couple README’s of the repository but I feel it is necessary to mention it here. If you have already followed these README’s and gotten Anaface to successfully build, you can safely skip this section and go to page 3.

## Project Settings

1. Go to “General” in the properties dialog for Anaface.
2. Make sure the Settings align properly (“Windows SDK Version” and “Platform Toolset” are two fields to check)

## Include Settings

1. Go to “VC++ Directories” 🡪 “Include Directories”
2. Remove the Folders currently listed in the sub-dialog
3. Reinsert the directories of “VideoGraphics” and “TrecLib” according to your cloned installation

## Library Settings

1. Go to “VC++ Directories” 🡪 “Library Directories”
2. Remove the Folder currently listed there
3. Reinsert the “Debug” folder so that it points to your debug folder

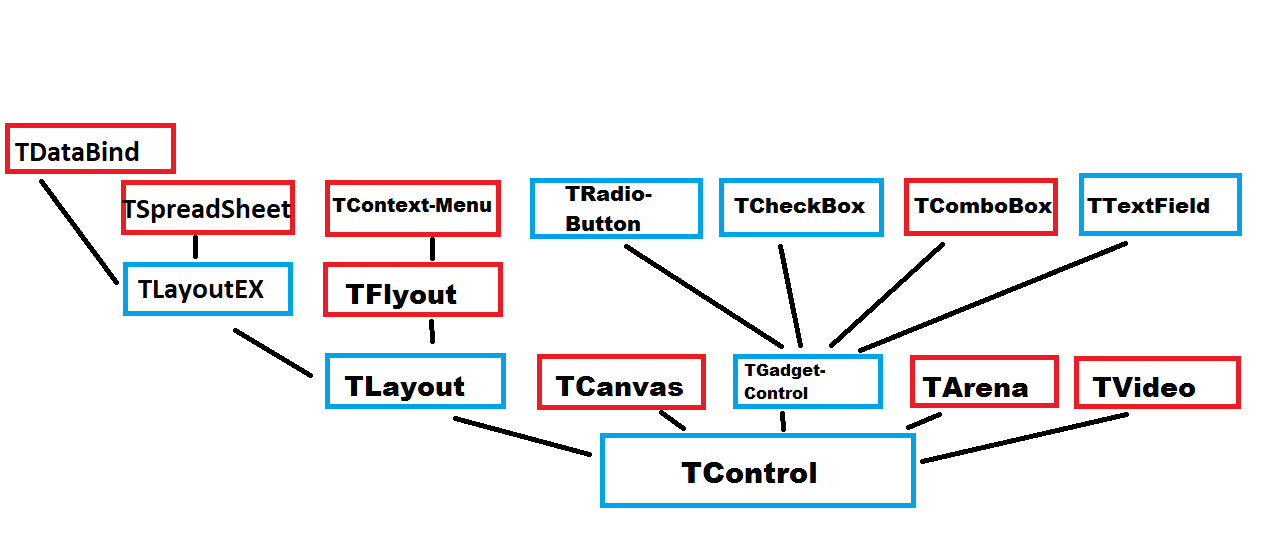
## Lib Files

1. “Linker” 🡪 “Input” 🡪 “Additional Dependencies”
2. Make sure that the following lib files are included
   * dwrite.lib
   * d2d1.lib
   * TrecLib.lib
   * VideoGraphics.lib
   * evr.lib
   * mf.lib
   * mfplay.lib
   * mfreadwrite.lib
   * mfuuid.lib
   * mfplat.lib
   * dxva2.lib
   * Strmiids.lib

# Class Layout

The core class involved in Anaface is the TControl. Any class that presents itself as a control that renders to an AnaGame screen and responds to User input derives from Tcontrol. The TControl has numerous fields involved but the most important fields are:

* 3 TBorder’s – classes that draw the border of the control
* 3 TContent’s – classes that render the body of the control (includes basic image support)
* 3 TText’s – classes that render text to the control
* Location – rectangle that represents the location of the control on screen
* Snip – rectangle that represents where the control can draw itself on screen
* Attributes – String map of attributes used by onCreate to set the other attributes
* Children – list of TControls that are the “children” of the given TControl



TControl: Root control, provides support for basic Event handling, location/snip management, Text/Border/Content management, and UI hierarchy.

TCanvas: Meant to be the “Drawing Board” of Anaface

TArena: Serves as the Camera for an ArenaEngine (see VideoGraphics library)

TVideo: Meant to play videos

TLayout: Organizes its children into rows, columns, or both

TLayoutEx: extends TLayout to support adding columns and rows

TSpreadSheet: meant to support Excel-like cells in a layout

TDataBind: Attaches to an Array of TObjects and add/removes rows based off of the contents of that array

TFlyout: Meant to operate outside the traditional hierarchy of TControls that appear when code calls for them to appear

TContext-Menu: Meant to appear when the user right-clicks

TGadgetControl: meant to allocate a space in the content for another drawing unique to subclasses

TRadioButton: Serves as the Radio Button of Anaface

TCheckBox: Serves as Anaface’s check-box

TTextField: Supports more complex text control, accepting keyboard input, and offers basic formatting, password and number management

TComboBox: Holds a drop-down menu that selects an option

# Building an Anaface UI

There are currently 2 ways to build an Anaface UI: hard-coding the structures, and using TML. Hard-coding them is inadvisable and can lead to messy code so that method will not be covered. TML, on the other-hand, is a mechanism for building Anaface Interfaces and will be covered.

1. Application identifies the TML file it wishes to use for its Anaface
2. Application opens up a TFile (TrecLib) of that file.
3. Application creates an AnafaceParser (Anaface) and provides it the following parameters
   1. A Direct2D render target – used by the controls to draw themselves
   2. A Window Handle – for controls that need to actually access the window itself
   3. A TString representing the directory (helps with referencing files with a relative path)
4. Application creates a TML\_Reader\_ (In TrecLib) with the Parser and file from steps 2 and 3.
5. Application calls the Read method of the Reader object.
   1. Reader parses the TML file.
      1. New Objects are added to the tree
      2. Attributes are added to the *current* object, which in turn stores these attributes in a String map (called *children*)
   2. If an error occurs during parser, the *Read* method returns false (setting the line number on which it occurred)
6. If *Read* returns true, Application retrieves the root control generated (and can delete the parser/reader after this).
7. The root control’s *onCreate* is called with the screen area the control is allowed to take.
   1. The control checks for various attributes and sets its actual attributes based off of these String attributes and the screen-space it was provided
   2. The control calls on Create for its children
8. The application is now free to call *onDraw* on the application

# Attributes in AnaGame

Note: These are attributes used in the Anaface library. Some are provided in the core TControl. Others are provided by sub-classes with more specific functionality.

|  |  |  |
| --- | --- | --- |
| **TControl Structure attributes** | | |
| **TML Attribute** | **Values** | **Description** |
| “VerticalScroll” | “True” | Used to force Control to generate a vertical scroll-bar |
| “HorizontalScroll” | “True” | Used to force Control to generate a horizontal scroll-bar |
| “Margin” | 4 comma-separated integers | Used to pad the control’s location from it’s boundaries |
| “Height” | Integer | The original height of the Control |
| “Width” | Integer | The original width of the Control |
| “MaxHeight” | Integer | The Max height the control is allowed to be |
| “MaxWidth” | Integer | The Max height the control is allowed to be |
| “MinHeight” | Integer | The Minimum height the control is allowed to be |
| “MinWidth” | Integer | The Minimum width the control is allowed to be |
| “FixedHeight” | “true” | False by default, whether the control ever changes its height |
| “FixedWidth” | “true” | False by default, whether the control ever changes its width |
| “Shape” | “Ellipse”, “RoundedRectangle”, “Custom” | Sets the shape of the control (default is rectangle) |
| “RoundedRectX”/ “RoundedRectY” | float | Sets the degree of roundedness of the control 🡪 only used if “Shape” is set to “RoundedRectangle” |
| “Class” | String (‘;’ separated) | Allows TControl to adopt the attributes of a certain *class* of control (similar to how it works in HTML) |
| “id” | String | Allows control to have a name (similar to HTML’s *id*) |
| “ContainerLoc” | 4 comma-separated integers | Used to set boundaries for child controls (set this attribute multiple times). Note: For TLayout and sub-classes, use their mechanism for organizing child controls |
| “FlyoutLocation” | 4 comma-separated integers | Used for any Flyout Control attached to the current control |

## TControl

Provides location management, size management, basic Text/Border/Content management, control trees, and basic event management.

Note: This is being provided in TWO tables: a Structural Table and a Style Table

|  |  |  |
| --- | --- | --- |
| **TControl Style Attributes** | | |
| **Text Styles** | **Border Styles** | **Content Styles** |
| **TML Attribute** | **Values** | **Description** |
| “BorderThickness” | Float | The Thickness of the border (1.0f default) |
| “BorderColor” | 4 comma-separated floats | The Basic Color of the border |
| “BorderColor2” | 4 comma-separated floats | An optional second color of the Border. If not provided, the “BorderGrad…” attributes have no effect and brush style defaults to solid. |
| “BorderGrad1”/ ”BorderGrad2” | Float | The Gradient *push* of each color, used by Direct2D and DirectWrite |
| “BorderGradMode” | “Radial” | How a two-colored Border would look. Defaults to “Lateral” if not provided (or “Solid” if text has 1 color) |
| “ContentThixkness” | Float | ? |
| “ContentColor” | 4 comma-separated floats | The basic color of the content area |
| “ContentColor2” | 4 comma-separated floats | An optional second color of the Content Area. If not provided, the “ContentGrad…” attributes have no effect and brush style defaults to solid. |
| “ContentGrad1”/ “ContentGrad2” | Float | The Gradient *push* of each color, used by Direct2D and DirectWrite |
| “ContentGradMode” | “Radial” | How a two-colored content area would look. Defaults to “Lateral” if not provided (or “Solid” if text has 1 color) |
| “ImageSource” | String – file path relative to TML file | Allows an Image to be presented in the content area (if the image is available) |
| “Caption” | String | What the text actually says |
| “CaptionLocale” | String | The locale of the text (default “en-us”) |
| “Font” | String | The Font of the text |
| “FontColor” | 4 comma-separated floats | The basic color of the font |
| “FontSize” | Float | The Size of the Font |
| “HorizontalAlignment” | “Left”, “Right”, “Justified” | The horizontal alignment of the text (Default is Center) |
| “VerticalAlignment” | “Top”, “Bottom” | The Vertical alignment of the test (Default is center) |
| “TextColor2” | 4 comma-separated floats | An optional second color of the text. If not provided, the “TextGrad…” attributes have no effect and brush style defaults to solid. |
| “TextGrad1”/ “TextGrad2” | Float | The Gradient *push* of each color, used by Direct2D and DirectWrite |
| “TextGradMode” | “Radial” | How a two-colored text would look. Defaults to “Lateral” if not provided (or “Solid” if text has 1 color) |

Note: A TControl can have Three Text components, three borders, and three content objects. The second set of components activate when the mouse hovers over the control. The third set of components show when the user clicks of the control.

To set the attributes of the second set of components, Simply prepend “Hover” before each style attribute. To set the third set of components, prepend “Click” before the same attributes.

## TGadgetControl (extends TControl)

|  |  |  |
| --- | --- | --- |
| **TGadetControl Structure attributes** | | |
| **TML Attribute** | **Values** | **Description** |
| “BoxSize” | Integer | The Size of the Small Box |

Meant to provide a mechanism through which a small button can be added to the side

## TArena (extends TControl)

|  |  |  |
| --- | --- | --- |
| **TControl Structure attributes** | | |
| **TML Attribute** | **Values** | **Description** |
| “EngineID” | String | The Name of the 3D “Engine” this control is going to work with |
| “CameraType” | “LookAt” | Whether the camera is looking AT something or in a Particular location (Default is look to) |
| “StartingDirection” | 3 comma-separated floats | Where in the 3D space is the camera looking to (if camera is *LookTo*, then these coordinate will be normalized) |
| “StartingLocation” | 3 comma-separated floats | Where in the 3D space is the camera positioned |
| “Up” | 3 comma-separated | 3D direction to consider *Up* (usually 0.0, 1.0, 0.0) |

Control used to interact with the VideoGraphics library and integrate AnaGame’s 3D library with Anaface. It is the job of this control to provide the engine with Camera information prior to 3D drawing

## TVideo (extends TControl)

Note: an attempt to support video playback was made a couple years ago. However, the effort fell flat. I opted to abandon this functionality (for now) in favor of other portions of AnaGame, such as other Anaface controls, 3D graphics, and other aspects.

Whatever code is there, you are free to implement your own playback mechanism. As long as it can be extended to support multiple codecs, much of the engine is done in the VideoGraphics library, I’ll be open to incorporating it.

## TRadioButton (extends TGadgetControl)

Used to support Radio Buttons (the buttons that can be grouped in such a way that only 1 button in a group is *clicked* at a time.

The Radio Button does not introduce any new attributes. They do, however, use the *class* attribute in *TControl* to group themselves.

|  |  |  |
| --- | --- | --- |
| **TControl Structure attributes** | | |
| **TML Attribute** | **Values** | **Description** |
| “BoxEntry” | String | The contents of the entry |
| “SubHeight” | Integer | The height of each sub-entry when they are shown |
| “DefaultText” | String | The text to initially display in the main control |

## TComboBox (extends TGadgetControl)

Adds basic Combo-Box functionality to Anaface. At this point, it only supports static contents set during Anaface construction.

## TCheckBox (extends TGadgetControl)

Does not introduce any new attributes at this time.

## TTextField (extends TGadgetControl)

|  |  |  |
| --- | --- | --- |
| **TControl Structure attributes** | | |
| **TML Attribute** | **Values** | **Description** |
| “IsPassword” | String | Should the control mask the contents? |
| “PasswordPeek” | Integer | If the contents are masked, should there be a button to show the contents? |
| “CanEdit” | String | Whether the User can enter input into it through the keyboard |
| “IsNumberControl” |  | Whether to only hold numbers |
| “DrawNumberBoxes” |  | Whether or not to draw number boxes on Fields set to hold numbers (defaults to true if *IsNumberControl* is true) |
| “Minimum” | Integer or Float | The minimum value a Number Field can hold |
| “Maximum” | Integer or Float | The Maximum value a Number Field can hold |
| “DefaultIncrement” | Float or Integer | How much to change the value through the Number boxes |

Meant to provide more complex Text manipulation functionality as well as basic input support.

## TLayout (extends TControl)

Organizes Child elements into Rows, Columns, Or both. The number of Rows and/or columns are predetermined and static, but control does support rows/columns of both fixed and flexible dimensions.

|  |  |  |
| --- | --- | --- |
| **TControl Structure attributes** | | |
| **TML Attribute** | **Values** | **Description** |
| “ColumnWidth” | Integer[\*] | Adds a new Column of a given width at construction (add a ‘\*’ to enable the column to have a flexible width) |
| “RowHeight” | Integer[\*] | Adds a new Row of a given height at construction (add a ‘\*’ to enable the row to have a flexible height) |
| “InternalBorderColor” | 4 comma-separated floats | The Color of the internal border |
| “InternalBorderThickness” | FLoat | Sets the Thickness of internal borders |

## TFlyout (extends TLayout)

|  |  |  |
| --- | --- | --- |
| **TControl Structure attributes** | | |
| **TML Attribute** | **Values** | **Description** |
| “Type” | “TCanvas”, “TGrid”, “TStack” and “TGallary” | What type of content is the Flyout likely to hold? |
| “RowHeight” | “Click”, “Hover”, “R”, “Click\_Hover”, “Click\_R”, “Hover\_R”, “Click\_Hover\_R”, “ScriptOnly” | When should the flyout appear? |

## TContextMenu (Extends TFlyout)

|  |  |  |
| --- | --- | --- |
| **TControl Structure attributes** | | |
| **TML Attribute** | **Values** | **Description** |
| “SubHeight” | Integer | The height of each option in the menu |
| “ContextMenuItem” | String | Contents of a given selection |

## TLayoutEx (extends TLayout)

The TLayoutEx does NOT add any attributes to the TLayout control. It does, however, add the ability to add or remove columns and rows more dynamically

## TSpreadSheet (extends TLayoutEx)

|  |  |  |
| --- | --- | --- |
|  | | |
| **TML Attribute** | **Values** | **Description** |
| “HeaderWidth” | Integer | How many columns the header should span |
| “HeaderHeight” | Integer | Height of the Header cell |
| “HeaderCaption” | String | What to hold in the header cell |
| “RowCount | Integer | Number of rows to hold |
| “ColumnCount” | Integer | Number of columns to hold |
| “SplitCharacters” | String | How to split the data in *SheetData* |
| “SheetData” | String, with a separator specified by *SplitCharacters* | Initial data to use |
| “DrawLines” | “False” | Whether lines should be drawn between cells (Defaults to true) |
| “NumOnly” | “True” | Whether the cells should hold only numbers (Defaults to false) |

## TDataBind (extends TLayoutEx)

|  |  |  |
| --- | --- | --- |
| **TDataBind Attributes** | | |
| **TML Attribute** | **Values** | **Description** |
| “DataDirection” | “Row”/”Horizontal” | How many columns the header should span |

# To-Do:

1. Have Location and Snip held by only TControl and have TControl pass references to each Border/Content/Text component upon draw
2. Add curve drawing support (should help with scrolling ellipses
3. Improve Scrolling Functionality and subject it to extreme tests
4. Improve Border Styling to make CSS implementation easier
5. Improve TTextField’s ability to control the caret.
6. Improve TTextField’s ability to cut/copy/paste
7. Add additional tags to HTML Parser
8. Add compilence with CSS standards
9. Fix TDataBind’s data management (should do once Scrolling is improved)
10. Make TContextMenu more flexible and able to have functional options
11. Add some functionality to TCanvas

# My To-Do:

1. Add Document layer to support the DOM interface in JavaScript (or any programming language AnaGame ends up supporting.