AnaGame Overview

|  |  |
| --- | --- |
| Stats | |
| Programming Language: | C++ |
| Compiler: | Visual C++ |
| Starting IDE: | Visual Studio 2015 Community |
| Current IDE: | Visual Studio 2019 Community |
| Dev Operating System: | Windows 10 |
| Intended Minimum Supported OS: | Windows 7\* (this was never tested) |

AnaGame is a development and application platform currently being developed for Microsoft Windows. Hopefully, it is eventually ported to Unix-like systems such as Apple’s OS X and Linux – though none are in development and no plans are being made at this time.

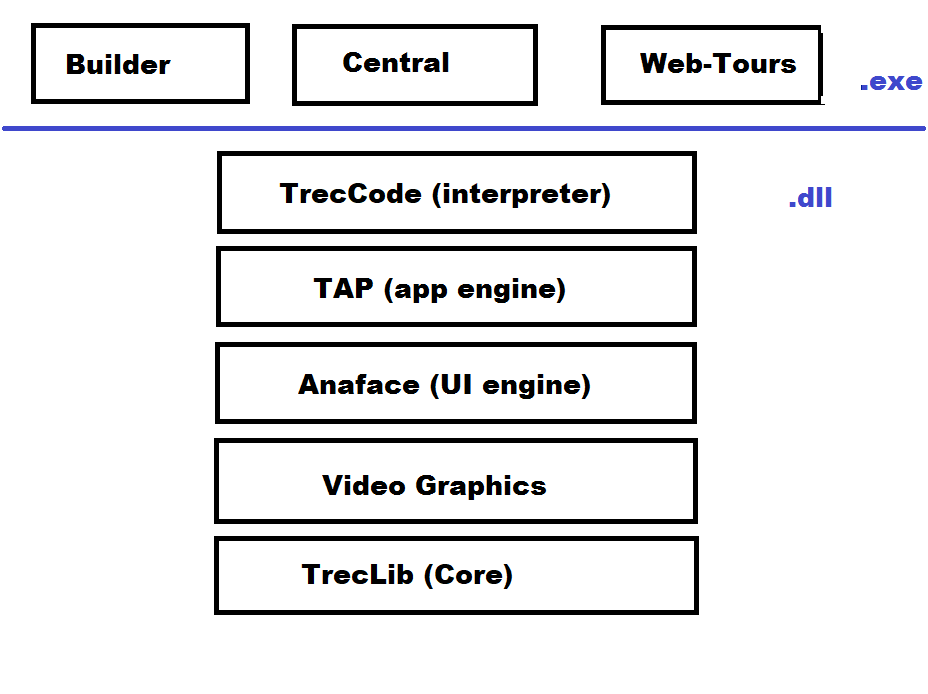
AnaGame is divided into different Projects, most of which interact with each other in some way shape or form (except MFCLibrary1, I need to remove that at some point). There are three major applications planned for final release and an additional *scaffold* application, each of which rely on a stacked structure of MFC Extension DLLs. Extension DLLs are libraries that export whole classes for higher-up projects to use.

|  |  |  |
| --- | --- | --- |
| DLLs | | |
| Library | Status | Description |
| TrecLib | Used | Provides the most basic services available in AnaGame such as the root object, TrecPointers, and other basic classes such as TStrings, TFiles, Parsers and special file readers |
| VideoGraphics | Used | Provides 3D rendering support (and eventually Video playback support), managing shaders, 3D models, and the environment in which they exist in |
| Anaface | Used | Provides the UI engine and the various controls that render themselves on AnaGame windows |
| Tap | Used | Provides higher-level resources such as TDialogs, pages that represent a screen for Anaface to render to, as well as wrapping everything up in an AnaGame program (not done yet) |
| TrecLib | In Dev, not Used yet | Runs on top of the Rest of the API and provides generic compilers and a generic interpreter that can access AnaGame resources |
| AnafaceSupport | On Hold (not Extension DLL) | Meant to be a functionally driven library with the intention of providing the means to support AnaGame control attributes AND CSS attributes. |
| |  |  | | --- | --- | | AnagameCompiler |  | | Deprecated, functionality replaced by TrecLib | Meant to be a functionally driven Library holding a compiler for AnaGame |

Functional as these Libraries may be, they cannot run on their own. This is where applications come in. There are four applications in AnaGame, three of which are supposed to be in the final product.

|  |  |  |
| --- | --- | --- |
| AnaGame Applications | | |
| Title | Status | Description |
| AnimateCentral | On Hold | Meant to be the hub application that runs AnaGame programs and plays videos. Currently used to test Anaface controls |
| Web\_Tours | On Hold | Meant to be AnaGame’s dedicated Web Browser – a project whose development spurred Anaface’s support for HTML/CSS (support, I admit, is pitiful) |
| AnimateBuilder | On Hold, Scaffold | Meant to serve as the IDE for AnaGame. It is more MFC based than AnaGame based and should be used to develop new features in isolation to one another.  It is NOT meant to be in the final product |
| AnaGame\_Builder | In Dev | Meant to serve as the IDE for AnaGame. It uses MFC as a base structure but uses AnaGame resources for everything else. |

The following image showcases AnaGame’s general structure. It’s a rudmitary image of what AnaGame looks like from a structural perspective (TrecCode was added recently) and there is only one Builder though the repository has two “Builders”.



# “Animate” or “AnaGame”

The Project is currently called *AnaGame*. However, when it was first started, the project was *Animate*, hence the solution name and some of the projects having the *Animate* name. This is similar to the tendency to change name mid-production.

For instance, Microsoft began working on a Windows API wrapper called AFX. However, they decided to call it MFC before releasing it to the public. That is why MFC often uses “afx” in their file names and other resources.

# Memory Management and the TrecPointer

TrecPointers are smart pointers used by AnaGame to manage memory. The smart pointer utilizes a Table to hold actual references to objects. Smart Pointers in general use a counter and delete the object once the counter reaches 0.

Originally, AnaGame used raw pointers. However, memory concerns (chiefly memory leaks) prompted the use of smart pointers. Initially, the std::shared\_ptr from the Standard C++ Library was used. However, there was the issue that occurs in this piece of code.

int\* ptr\_i = new int(3);

std::shared\_ptr<int> sh\_ptr\_i1 = ptr\_i;

std::shared\_ptr<int> sh\_ptr\_i2 = ptr\_i; // Error

std::shared\_ptr<int> sh\_ptr\_i3 = sh\_ptr\_i1;

The way the creation of the second pointer is an error is that there are now two smart pointers with 2 separate counters referencing the same object. Therefore, if one of those smart pointers is assigned something else (or when the first one goes out of scope), the counter will reach 0 and the object will be deleted – even though there is still another smart pointer referencing it.

(The final line works as the two smart pointers share the same counter as well).

Therefore, AnaGame provides the TrecPointer, a special type of smart pointer where pointing to the same object means using the same counter as well. This is because TrecPointer’s don’t allow you to assign raw pointers to them. In order to instantiate an object into a trecPointer, you need to use the *TrecPointerKey* class.

int\* ptr\_i = new int(3);

TrecPointer<int> sh\_ptr\_i1 = ptr\_i; // Compile Error: cannot assign via Raw Pointer

TrecPointer<int> sh\_ptr\_i2 = TrecPointerKey::GetNewTrecPointer<int>(3); // Correct!

More information on the TrecPointer will be available shortly – including an in-depth discussion on how they work, where the references are stored, and what type of error checking mechanisms are in place. However, because they are an important topic in AnaGame development and this is likely the first Document on AnaGame you have read, it makes sense to mention them here.

## TrecPointer rules

1. Do NOT have an Object pointed referenced by both a TrecPointer and an std::shared\_ptr.
   * While TrecPointer is preferred, if you choose to use a different smart pointer, stick with it.
   * There is NO mechanism through which the TrecPointer's counter can be synced with that of an external smart pointer
2. If a given Object derives from IUnknown, i.e. you're working with a DirectX/COM object, use the TrecComPointer instead
   * COM objects play by a different set of rules than the usual new/delete set.
   * You’ll need to use the *TrecComHolder* and its methods to instantiate an *IUnknown*.
3. Before using an object held by a TrecPointer/TrecComPointer, perform a NULL check.
   * Simply call it's .Get() method within an if statement

# How DLL Resources are exported/imported

AnaGame takes advantage of C/C++’s Preprocessor and its ability to set Macro’s. While I’m not going into the details over how resources are exported across projects, I will briefly talk about how AnaGame manages DLL importing and exporting.

// In TrecLib.h

#pragma once

#ifndef \_TREC\_LIB\_DLL

#define \_TREC\_LIB\_DLL \_\_declspec(dllexport)

#else

#define \_TREC\_LIB\_DLL \_\_declspec(dllimport)

#endif

Looking at the TrecLib project (for example, each DLL project has a file like this), a single macro is set as either an export macro or an import macro. When TrecLib is being compiled, the macro is set to export. If a different project is being compiled, the macro becomes an import.

Therefore, any resources (class, struct, union, or function) intended to be visible outside the library needs to be marked with this macro (or the project’s comparable macro).

# To-Do’s:

Obviously, AnaGame is far from complete (at best, I’d call it Version 0.2). There are three categories of To-Dos:

1. My-Do’s: Aspects of AnaGame I’m currently working on. Although you’re welcome to work on those aspects as well, we may arrive at completely different implementations.
2. Generic To-Do’s: Changes to AnaGame that could be made by anyone currently. These To-Do’s, I’m not currently focused on right now.
3. Eventual To-Do’s: Changes that are on the horizon, but require other To-Do’s to be fulfilled before tackling them makes sense.

Details of each To-Do will be provided in Specific documentations for each library, including relevant classes, methods, and functions (plus which library they are found in).

## My To-Do’s:

1. Flesh-out the Interpreter in TrecCode
2. Improve the TDataBind and TSpreadsheet controls in Anaface.
3. Continue the ArenaApp in AnaGame\_Builder and the dialogs, controls being developed there.

## Generic To-Do’s:

1. Add a Matching class that handles Regular Expressions
   * Should likely be placed in the TrecLib library
   * I envision it holding a TString that is a regular expression and report whether is provided string conforms to that regular expression
2. Networking support
   * Not sure how to properly approach this: all functionality in TrecLib, a base class in TrecLib and subclasses in a separate DLL, or All networking functionality in a separate DLL (likely between TrecLib and VideoGraphics)
   * Should ultimately support all types of Application layer protocols such as HTTP, HTTPS, FTP, SSH, etc.
3. Cryptography support
   * Needed if HTTPS is to be supported
   * Not sure if it should be in TrecLib, above TrecLib, or below TrecLib (yes, you read that right).
4. Video Support
   * Sometime ago, I attempted to implement a Video player using Microsoft Media Foundation and using a tutorial.
   * Although the TPlayer (VideoGraphics) and TVideo (Anaface) appeared to be working properly, I was never able to get a video to play.
   * Not sure if AnaGame should stick with Media Foundation, use Direct3D, a third party library, or some complex mechanism of its own
5. Anaface scrolling
   * Implemented a basic scrolling mechanism some time ago
   * Only subjected it to basic scenarios, will likely fail more complex scenarios.
   * Need to have controls recognize whether or not they need scrolling
   * Many controls have no sense of whether to draw their whole selves or portions of themselves.
   * If controls happen to be ellipses or rounded rectangles, how to draw only sections of those
6. Improved HTML/CSS Support
   * Support for more HTML tags and better mechanism for handling span tags (which would not work properly under the current set-up
   * A mechanism to handle both Anaface attributes and standard CSS attributes – currently, you could use a CSS file, but they have to use Anaface attributes, not CSS
7. More Readers
   * Formats such as JSON, XML amongst others should be supported.
   * Readers should likely be in TrecLib.

## Eventual To-Do’s:

1. Complex shaders (VideoGraphics) that deal with a variety of 3D features such as lighting, normalization, textures, physics, etc.
2. GIF support
3. Drawing
4. Compiler Support
5. Wrapper functions around various Methods