### UI系统介绍

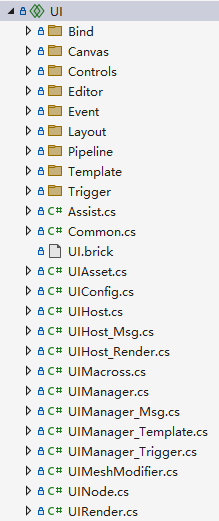
#### 代码目录结构

除绘制及文字处理外，其他代码均在C#实现

C++ 代码位于Cpp/Bricks/Canvas中

C# 代码位于CSharp/Bricks/UI中

绑定相关



配置相关，这个配置后面会在工程配置中显示，设置通用的UI配置，如DPIScale等

触发器相关

模板相关

管线相关，目前里面放了屏幕控件UI的相关代码，默认UI是3D UI

布局相关

事件相关

编辑器相关

控件相关

绘制相关

#### 控件结构

**TtUIElement**

**TtText**

**TtCheckBox**

**TtImage**

**...**

**TtButtonBase**

**TtContainer**

**TtUIHost**

**TtToggleButton**

**TtButton**

**TtCanvasControl**

**TtBorder**

* **TtUIElement**

所有控件的基类，包含了绑定、布局、消息、模板等一些控件的基本处理代码，所有的控件都从它或者它的派生类派生。

* **TtContainer**

所有容器的基类，提供了容器基本布局以及子控件的处理。容器中提供控件的布局操作，不同的容器有不同的布局模式，详细的布局信息会在布局部分说明。

由于有模板的存在，所以容器的父子关系分为两种，一种是可视化父子，一种是逻辑父子，TtUIElement.Parent及 TtContainer.Children获取到的是逻辑子，如果要获取可视化子，需要使用VisualTreeHelper.GetParent和VisualTreeHelper.GetChild

默认

Button

Text

显示模板

Button

Border

Text

示例：

左边TtButton是一个模板控件，下面添加一个TtText用于显示按钮上的文字

实际上模板中隐含了一个TtBorder作为TtButton的子，用于给TtText进行布局

此时

Button.Children[0]是Text

Text.Parent是Button

VisualTreeHelper.GetChild(Button, 0)是Border

VisualTree Helper.GetParent(Text)是Border

* **TtUIHost**

UIHost为一个窗口的根，每个在3D场景的UI窗口的均有一个UIHost作为根控件，它包含了窗口大小以及跟SceneNode的交互，ScreenSpaceUI则是在View中有属于自己的UIHost，保证在分屏游戏中每个view都有自己的屏幕空间UI。另外在UI编辑器中显示的UI的host是EditorUIHost，它从TtUIHost派生，包含了和编辑器的一些交互。

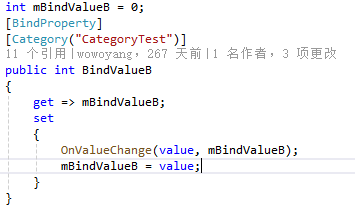
* **TtContentsPresenter**

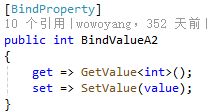
模板专用控件，用来标识此位置可以用来放置子控件，后续模板部分会有介绍。

控件类上打上Editor\_UIControlAttribute就可以在编辑器的控件面板里显示了。

#### 绑定

绑定部分主要依赖属性，只有打上BindPropertyAttribute的属性才能够进行绑定操作。绑定属性的get 和set有几种写法，分别是



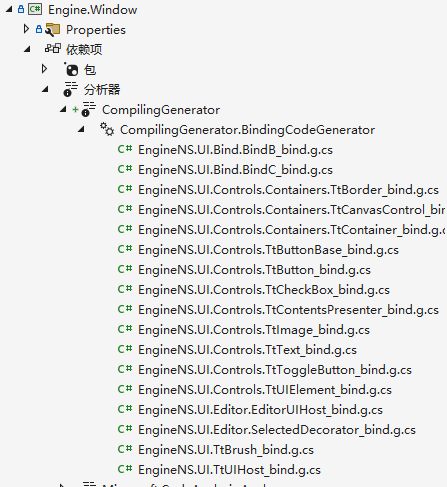


其中OnValueChange以及SetValue会通知绑定到这个属性的对象进行更新，如果不使用这两个函数则这个属性改变时无法通知绑定刷新，但是不影响别人改变时它刷新。

为了方便使用、不依赖反射以及减少GC，绑定部分的注册以及调度完全依赖代码生成，这部分的代码会在macross或者工程在编译阶段进行生成并参与编译，用于代码生成的代码在CompilingGenerator.BindingCodeGenerator中实现。

代码生成会生成属性对应的TtBindableProperty类型的静态对象方便UI系统及其他地方调用，名称为属性名称+Property，如BindValueB对应的是BindValueBProperty。

引擎生成的代码可以在这里看到和调试，游戏项目生成的代码不可见：

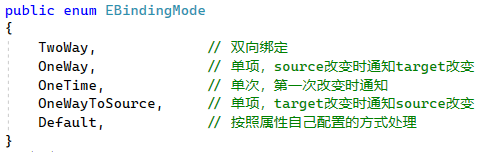


绑定的使用部分主要有两个函数：

TtBindingOperations.SetBinding：处理属性和属性的绑定

TtBindingOperations.SetMethodBinding：处理属性和函数的绑定

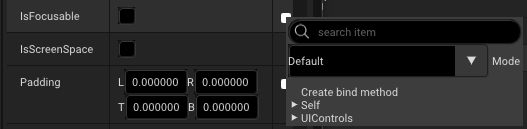
绑定模式如下：



另外在绑定时支持数据类型转换，需要从TtBindTypeConvertBase派生并实现对应接口。

绑定相关的示例代码可以看BindCommon.cs中BindA,BindB,BindC,Int2DoubleConver以及BindTestClass相关代码。

需要注意在编辑器中配置绑定时，如下图



IsFocusable作为Target，配置的绑定作为Source的。

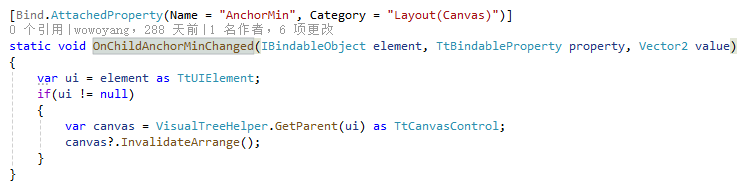
#### 附加属性

UI系统提供了附加属性的能力，作用是在控件放入不同容器时可以给控件提供该容器特有的属性访问。比如：

控件A加入Grid容器中，在编辑器中选中控件A，属性面板上会显示Grid附加给控件A的属性，比如Row，代码中可以通过Grid.GetRow(A)，Grid.SetRow(A)来获取或设置控件所在的行。

当把控件A从Grid容器中取出然后放入Canvas容器中时，Grid的相关数据失效，这时编辑器中选中控件A，可以在属性面板上看到Canvas容器附加给控件A的属性，比如AnchorMin，代码可以通过使用Canvas.GetAnchorMin(A)等函数来获取Canvas特有的数据。

附加属性可以通过在静态函数上添加AttachedPropertyAttribute来设置，示例：



上图中附加属性的名称是AnchorMin，这个属性会显示到编辑器的属性面板中，并且在该值改变的时候会调用OnChildAnchorMinChanged。

编译时会通过代码生成实现附加属性的注册以及调度处理。

#### 绘制

UI中对绘制的主要包装放到了TtBrush类中，包含了对UVAnim的处理和与底层绘制接口的交互，主要的逻辑在Draw函数中。

UVAnim有单独的资源编辑，这个和之前是一致的。

控件中有绘制需求的需要加入TtBrush，然后重载TtUIElement的Draw接口实现自己的绘制逻辑。具体可以看TtImage的实现。

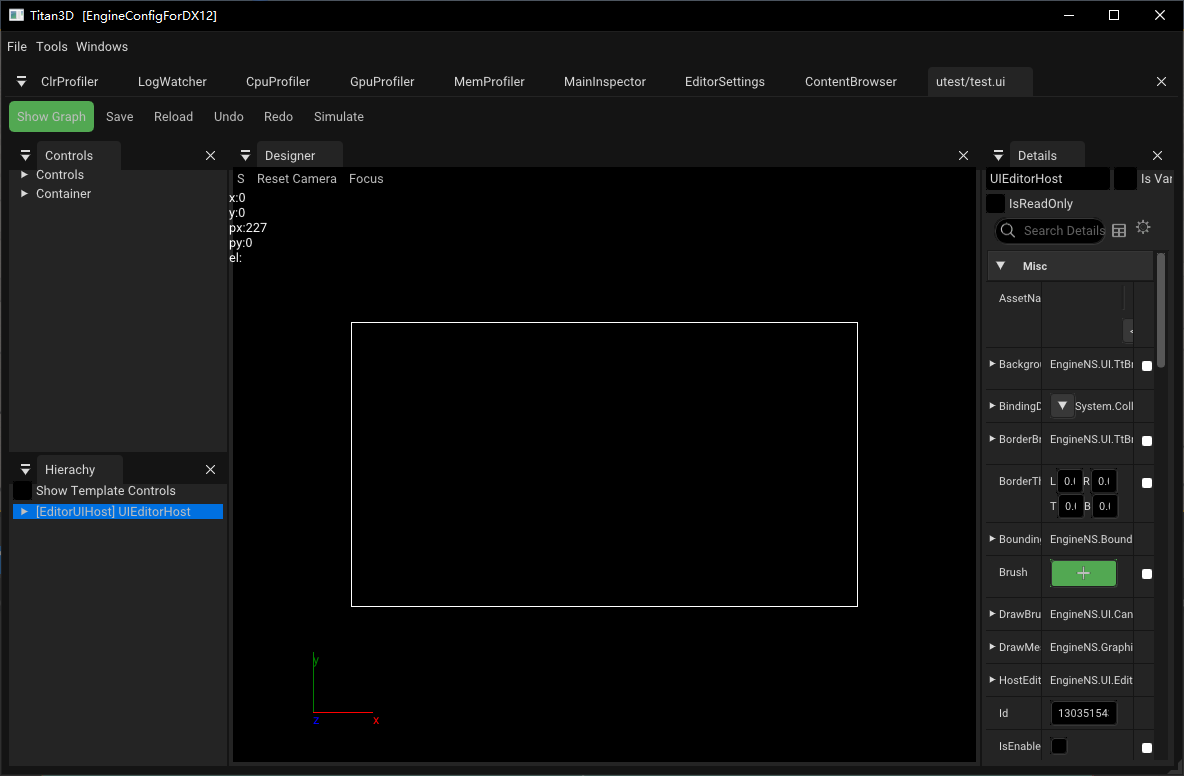
TtButton中三态按钮的实现是通过模板来做的，所以内部不包含绘制逻辑。具体模板部分会做说明。

TtUIElement中有一个属性RenderTransform，可以单独设置该控件的Transform以达到3D界面的效果，该参数在绘制过程处理，所以不会对布局产生影响。

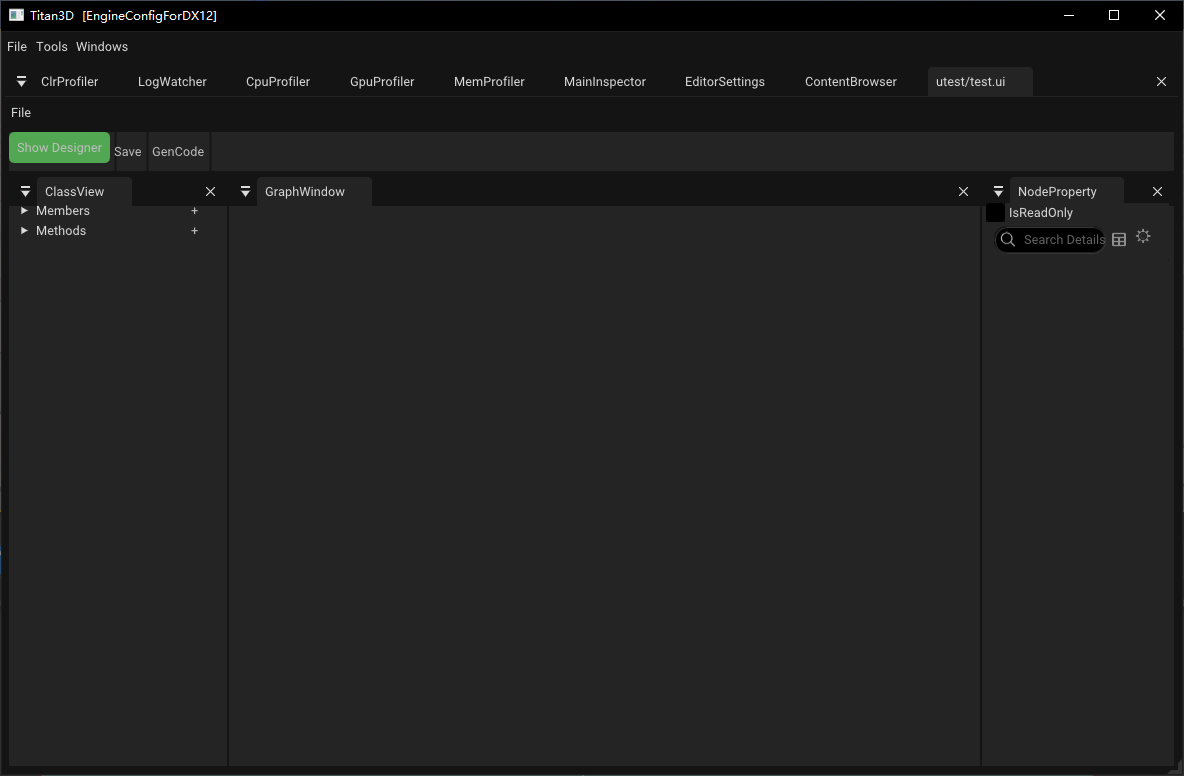
控件中如果有影响最终绘制结果的参数改变，需要手动设置MeshDirty为true，它会通知对应界面的UIHost设置MeshDirty，每帧UIHost都会检查这个参数，如果为true，则启动BuildMesh流程重新生成界面的Mesh。RenderTransform的改变不会启动BuildMesh的流程。

#### 编辑器

编辑器有两个部分组成，一个是Designer，用来编辑界面、设置属性、绑定等



另一个是Macross，用来给界面添加逻辑、按钮响应等



目前编辑器还有UITrigger部分以及模板创建没有实现，基本功能已完备。

由于容器有自己独特的布局数据和编辑方式，所以在编写容器时需要同时实现编辑器中对容器的编辑操作类，从IUIEditorDecorator派生实现相应的接口即可，可以参考Decorator\_Canvas

#### 事件

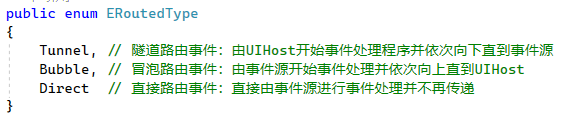
目前在TtUIELement中已经定义了基础的鼠标、触控以及键盘事件，其中包含跨输入事件，比如DeviceDown，它在MouseButtonDown和TouchDown响应时均会响应。

为方便后续扩展，事件采用注册的方式添加，下面是Button的Click事件：



其中TtEventManager.RegisterRoutedEvent用于往系统中注册事件，注册时需要

1. 设置事件类型



1. 设置HandleType

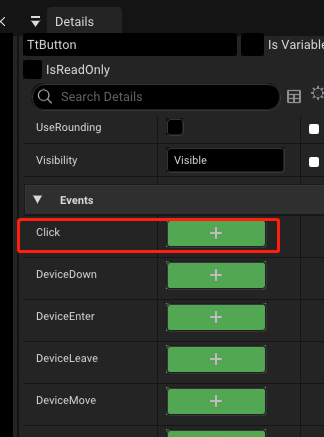
目前只实现了 TtRoutedEventHandler ，后续视情况添加

Click这个event用于方便外部添加事件响应相关代码，

例如：Button.Click += something.ClickAction;

OnClick是Button用于触发事件的代码，通过RaiseEvent来通知系统触发事件。

注册后的事件会在编辑器中的属性列表中显示，如下：



点击后会在Macross中自动创建一个函数用来响应事件

#### 布局

布局部分主线还是采用Measure+Arrange的方法，由TtUILayoutManager在每帧根据收集到的需要布局的控件进行布局处理。

当控件需要通知系统需要刷新布局信息时，可以调用UpdateLayout(),另外也可以直接调用InvalidateMeasure()或InvalidateArrange()来单独通知需要Measure或者Arrange，Measure的过程如果发现需要Arrange，系统内部会调用InvalidateArrange()。

控件中如果有自己计算布局的需求，可以重载MeasureOverride和ArrangeOverride两个接口，

MeasureOverride中输入的availableSize是在测量过程中由父级控件能够提供的大小，可能是(0,0)，也可能是窗口大小，也可能是无限大（比如ScrollView），所以处理的时候需要注意。

ArrangeOverride中输入的是前面经过布局系统计算过后的真正能够提供的大小。

容器控件支持自定义布局，目前在TtCanvasControl中实现了锚点式的布局，也是通过重载MeasureOverride和ArrangeOverride实现的，可以作为参考。

**部分布局参数：**

Thickness类型的参数标识的基本都是宽度信息，如下：

**内部控件**

**容器**

Bottom

Right

Left

Top

Padding

BorderThickness

#### 触发器

触发器部分目前实现了TtUIPropertyTrigger，可以实现当一个或多个属性成为特定值时，触发操作，这个操作可以是执行某些函数，或者是设置某些属性的值。目前只做了设置值的功能。

触发器的基类是TtTriggerBase，所有类型的触发器都从它派生并实现接口，触发器触发后设置值的处理需要从TtTriggerValue派生，用于实现不同类型值的处理，目前实现了TtTriggerSimpleValue<TProp>，提供了属性的触发实现，后续看需要扩展，比如Event之类的。

控件会通过代码生成来生成一些触发器相关的代码，用来减少GC和反射的使用。

#### 模板

模板提供了通过组合控件元来实现复杂控件的能力，并通过触发器以及绑定来实现复杂的控件行为。模板类的基类为TtUITemplate,一般控件的模板类是TtControlTemplate。系统默认的模板初始化代码写在各控件中，通过静态构造初始化，如TtButton. InitSystemDefaultTemplate。

比如：TtButton中只包含了基本的Click等事件处理，外观以及多态按钮等功能并没有直接在代码中写死，这些都是通过创建模板来完成的，代码如下：



创建TtControlTemplate并指定TargetType，标识此模板是TtButton类型

设置TtButton的属性的默认值，这里设置了Background和BorderBrush，能够设置的必须是可绑定属性。

之后添加模板控件，这里在模板中添加了两个控件，一个是TtBorder，用来给Button的子控件提供布局，这里可以根据需要换成不同的容器来实现不同的布局操作。一个是TtContentsPresenter，这个是模板专用控件，用来提供一个占位，并告诉模板这里会由外部设置的控件占用，举个例子：

如果模板的结构如下：

TtControlTemplate(Type is TtButton)

TtBorder

TtContentsPresenter

TtImage

在外部使用这个TtButton模板时如果给TtButton添加了一个Text子控件和一个CheckBox子控件，不显示模板时看到的会是这样（逻辑树）：

TtButton

Text

CheckBox

当显示模板时看到的会是这样（可视化树）：

TtButton

TtBorder

Text

CheckBox

TtImage

注意，一个模板不能同时放多个TtContentsPresenter

代码中 SetTemplateBindingValue 标识将控件的属性和模板的属性绑定，模板的属性取决于模板的TargetType设置，由于模板内控件对外部是不可见的，所以这里可以通过内部控件的属性与模板属性绑定来达到内部控件属性的控制，比如：



这里是将buttonRoot这个TtBorder的Background属性绑定到模板TtButton的Background属性上，这样在设置TtButton的Background的值时，TtBorder的值会跟着改变。

在设置Trigger部分，通过添加条件让按钮具备三态的功能。Trigger可以添加多个Condition来达到复杂的控制，另外在Trigger触发后条件不再成立时，会将改变的值恢复到改变之前。目前只实现了TtUIPropertyTrigger，后续还需要实现Event Trigger等。

#### Macross

每个UI界面都伴随一个Macross，用来进行基础的事件、绑定以及逻辑处理，Macross类型派生自TtUIMacrossBase，在UI编辑器保存时会生成代码，并保存在UI资源的同目录下，类名称与资源名称一致。

#### Manager

TtUIManager提供了UI的基础管理，并提供UI的保存、加载能力，所有其他系统与UI系统的交互都要通过TtUIManager来完成。Load接口加载后会返回加载的UI根容器，由外部代码控制该容器的处理，它不是UIHost类型的，因为加载出来的可能时窗口，也可能是控件。

#### 未完成的控件列表

目前UI的基本系统已完备，还差大量的控件来完善UI的功能，部分控件使用基础控件搭建模板来实现功能和外观。

控件列表如下：（绿色字的已完成，黄色字的制作中，黑色字的还没有实现，蓝色底的是基础控件，橙色底的是扩展控件，需要用基础控件组合模板来定义外观，灰色底的是容器控件。

基础控件和容器控件是相对比较优先的，有了这些控件就可以用来组合模板控件了）

|  |  |  |
| --- | --- | --- |
| 名称 | 类名 | 说明 |
| Image | TtImage |  |
| ContentsPresenter | TtContentsPresenter |  |
| Text | TtText | 还在制作中 |
| Rectangle |  | 显示方框或圆角方框的控件 |
| Rich text |  | 富文本框 |
| Text Input |  | 文本输入框 |
| Grid splitter |  | Grid控件的分割器 |
| Popup |  | 用于弹出窗口的控件，如Combox的下拉列表框 |
| Track |  | Scroll控件的基础单元，滑块下面的底，和Thumb控件组合成一个基本的Scroll控件 |
| Thumb |  | Scroll控件的基础单元，鼠标可以操作的滑块，和Track组合成一个基本的Scroll控件 |
| Button | TtButton |  |
| ToggleButton | TtToggleButton | 代码基本完备，还没有写模板 |
| CheckBox | TtCheckBox |  |
| Radio Button |  | 这个在考虑 要不要做，因为和ToggleButton功能类似 |
| Menu |  |  |
| Progress bar |  |  |
| Slider |  |  |
| List View |  |  |
| Tree View |  |  |
| Combo Box |  |  |
| Expander |  |  |
| Tab Control |  |  |
| Tab Item |  |  |
| Scroll bar |  |  |
| Scroll view |  |  |
| Tool bar |  |  |
| Border |  |  |
| Canvas |  |  |
| Grid |  |  |
| Uniform Grid |  |  |
| Stack panel |  |  |
| Wrap panel |  |  |
| Safe zone |  | 类似Unreal的Safe zone，标记一个安全区域来显示UI内容，用来适配不同的刘海屏、挖空屏等 |