Architecture and features MyGUI  
  
Widget ( Widget )  
  
The main building material is a GUI widget . A widget is a rectangular area , with some properties and logic of interaction with the user. Widgets support hierarchy to create more complex widgets , loading of XML and other features :  
• Support for the Alpha, management transparency.  
• Support alignment. Widget can be stretchable along with the parent , or bind to any parent faces .  
• Support of the mask ( mask is any picture ) , to control the mouse picking , ie an option to create widgets of arbitrary shape , for example round .  
• Support for arbitrary cursor when you hover over a widget , the cursor will take the form specified .  
• Support windows tips (tooltip). The widget not stitched windows tips , but the widget can tell when you need to show the window and when to hide it . Accordingly, it is possible to construct any difficulty windows prompts.  
• Support detach and attach in the hierarchy. This enables the construction of floating plates.  
• Independent hierarchy hierarchy of rendering jobs . For complex widgets , such as windows with client area (Window, ScrollView , etc.) or a pop-up widgets with children (ComboBox) use the logical children that do not coincide with the physical , however , they behave like real kids . For example, when you hide the ComboBox, hiding it in a drop-down list in the Window list of children are those children who we've added , although they are physically on the client widget.  
• Support for skins ( skins ) on the fly.  
• Support to change the style (a style of behavior) on the fly. Eg, any child can make a pop-up , it will not be clipped to the size of the parent , or overlapping , it will overlap with other children.  
  
  
SubWidget ( Sabvidzhet )  
  
In fact , the widget is virtual , it is not displayed and does not know how to do it . The widget has a list sabvidzhetov on whose shoulders and go all the display logic . Sabvidzhety is also rectangular area , but inside the widget , they can also have alignment and occupy any position within the widget . Sabvidzhety created through the factory and can be of different types . List of types and sabvidzhetov given in a particular skin. When we set the widget skin , it creates the appropriate sabvidzhety and listed widget.  
  
Sabvidzhety present in the core:  
• MainSkin - sabvidzhet indicating texture and rectangle in it, stretched in the entire widget .  
• SubSkin - sabvidzhet indicating texture rectangle and it can have an arbitrary position in the widget.  
• TileRect - sabvidzhet indicating texture rectangle and it may Tile yourself in any direction and at any size . Parameters are specified in XML.  
• RawRect - sabvidzhet providing access to their tops to control from the outside . Can be used to implement custom widgets.  
• RotatingSkin - sabvidzhet indicating texture and rectangle in it, is rotatable to any angle relative to an arbitrary point.  
• SimpleText - sabvidzhet to display text.  
• EditText - sabvidzhet to display text , text offset support , word wrap (WordWrap), selection and cursor position.  
  
So, what else can be done with sabvidzhetov . In fact, very much, for example :  
• Sabvidzhet draws text geometry rather than texture.  
• Sabvidzhet displaying figures or other complex geometry (Cairo, Flash, SVG).  
• Sabvidzhet texture mapping blitingom ( GUI can be implemented without the 3D API)  
• Sabvidzhet displaying complex data (eg output html or RTF)  
  
  
State ( State , State)  
  
Each has a list sabvidzhet State , State, a named set of data. By default, the State «default». In every State , you can specify different data for the texture coordinates offset text , text color , or any other arbitrary data .  
State used to display the different widget states , for example, take the button . In simple buttons usually 4 states :  
1. disabled - button is not available to the user  
2 . normal - the normal state of the button  
3 . over - hanging over the button Mouse  
4 . pressed - the button is pressed  
  
Each sabvidzhet describes the data for relevant State , and when the widget changes its state , it goes around sabvidzhetam , and sets the current State . For example, in Staten disable text usually make dark or gray and Staten pressed text is usually shifted down by one or two pixels.  
  
  
Layer ( Leer , level)  
  
Leer is an abstraction , it is an area where there may be one or a group of widgets. Just Guard rails may be empty. For simplicity, we can imagine that when we look at the monitor , we look at the top layer cake . Each layer of the cake will correspond to one of Leer . Handholds are responsible for the behavior and display widgets, which are attached to this Leer . Each rail has a type , and created through the factory , ie possible to create your own types and affect the display or behavior of widgets included in this lifeline . Handholds which are currently in use , specified in XML, may specify additional data for lifelines .  
  
Handholds present in the core:  
• OverlappedLayer - All widgets are Rutaceae in Leer , may overlap, can be lifted to the top of the widget by clicking on it .  
• SharedLayer - Rutaceae All widgets have a single layer , each new Rutaceae widget does not separate buffer. This layer is used for optimization , the main application is a widget that never intersect , such elements HUD.  
  
Handholds can be used for other tasks, such as :  
• Otrendrit widgets present in Leer in texture , and then apply this texture on a 3D object.  
• Otrendrit Rutaceae individual widgets in texture and animate.  
• Otrendrit Rutaceae individual widgets in texture and show it instead of the widget ( caching ) .  
  
  
Controller ( Controller )  
  
Controllers are objects that are used to change the properties of any widget depending on the elapsed time. Controllers have different types and created through the factory that allows you to create arbitrary controllers . It is possible to specify in the XML required for controllers of any widget. There is support in the editor.  
  
Controllers present in the core:  
• ControllerPosition - produces a change in the position of the widget according to the specified function.  
• ControllerFadeAlpha - alpha changes .  
• ControllerEdgeHide - removes the widget from the window edge .  
  
You can use this to change any property widgets , everything is limited by your imagination . For example a smooth change of position in ProgressBar, by letter header output to simulate printing etc.  
  
  
Resourse ( Resource)  
  
All data which are running GUI resources . Resources are of different types , it is permitted to create their own types. Resource is an object that stores data in itself .  
  
Resources present in the core:  
• ResourceImageSet - description of a set of images with animation support and groups.  
• ResourceSkin - description of the skin ( rind) for the widget.  
• ResourceImageSetPointer - description of the cursor with animation support .  
• ResourceManualPointer - describes a simple cursor with texture and a rectangular area in it.  
• ResourceManualFont - font description with texture and coordinates indicating characters in it.  
• ResourceTrueTypeFont - font description to generate freetype library.  
  
Some resources are used in the core via an interface, it enables one for writing their resources, such as the generation of the font library or the other of the cursor using a more complex animation.  
  
  
Plugin ( plugin )  
  
The plugin is a wrapper for the code that allows independent distribution, use and connection on the fly any parts subsystems. Plugin can contain :  
• Custom widgets  
• Custom sabvidzhety  
• Custom handholds  
• Custom Controllers  
• User Resources  
• Independent from the GUI code  
  
Of course , the use of plug-ins are not mandatory , all its subsystems and other code can be directly in the common code , but using the plugin opens up interesting possibilities :  
• Use custom subsystems and types in the editor.  
• Reduce swelling of the main code .  
• Easy replacement , removal or modification of subsystems and types.  
  
  
LanguageManager  
  
GUI supports replacing arbitrary tags in rows and resources. The tag looks like this: # { tag\_name } . Replacement tags made ​​the downloaded dictionary. When downloading forms ( layouts ) replacement headers automatically. For example, to change the interface language is necessary , remove the window , load the proper vocabulary to create a window . After this replacement will be in the headlines of the new vocabulary . Replacing supported resource that allows you to make some semblance of order prescribing tags instead of the names of textures, colors and more skin or pictures.  
  
  
Delegate ( delegate )  
  
The delegate is a function pointer to an object wrapped . It makes function calls type-safe and simple. Eliminates the need for maps, tables and callback interface . Delegate maintains a pointer to the following types of functions :  
• The global (static) function .  
• Static class method.  
• Method object.  
• Virtual method object.  
  
  
other Features  
  
• Independent render. There is the possibility of writing render other platforms or for modification of existing essential purposes, such as optimization.  
• Support for Unicode - UTF8, UTF16, UTF32  
• Supports Drag & Drop  
• Clipboard support , W32 used in the system clipboard .  
• Supports color tag in HTML, such as the derivation of the line «# 0000FFblue», will be written «blue» blue.  
• Fast system RTTI.  
• An abstract data type . You can use custom data streams such as zip archives.  
• Support for versions of the resource types in XML. To maintain the older formats .  
  
  
External tools and components  
  
• Form Editor (LayoutEditor) to facilitate the design.  
• Generator for freetype font resource library.  
• Simple viewer resources pictures ImageSet.  
• Generator vraperov used to generate Managed C + + and C # wrappers . It is also possible to configure other languages.  
• Set Demo to demonstrate the capabilities and methods of use GUI.  
• Set UnitTest to test some critical parts or for the demonstration of complex techniques.