uViLEd

Visual Logic Editor

Last edition 04.07.2018

API

Introduction

Welcome to **uViLEd** programming guide. This document contains the description of all the classes, structs and enumerations, necessary for the development of components and their visual representation in the logic editor.

All API is divided into two big subsections: API for the development of components code and API for the development of components visual representation in the logic editor.

uViLEd Core

Core

This namespace contains the classes, implementing the core of uViLEd system, responsible for logics initialization and execution.

GlobalEvent

This class implements the global messaging system, based on data types. It's a singleton based on **MonoBehaviour.**

Variables:

Instance – the static instance of **GlobalEvent** class.

Public methods:

```
void Subscribe(object eventContainer)
```

Subscribe a container (class), by auto collecting all handlers, described in it.

```
void Subscribe<T>(Action<T> handler) where T : BaseEvent<T>
```

Subscribe the handler ***handler*** on event with data type T.

```
void Unsubscribe (object eventContainer)
```

Unsubscribe a container (class) from all events, having handlers inside it.

```
void Unsubscribe<T>(Action<T> handler) where T : BaseEvent<T>
```

Unsubscribe the handler ***handler*** from event with data type T.

Classes

BaseEvent

Generic base class for an event definition.

Example:

```
public class MyEvent : GlobalEvent.BaseEvent<MyEvent>
{
    public int EventData { get; private set; }

    public MyEvent(int data)
    {
        EventData = data;
    }
}
```

Static methods:

```
void Call(T data)
```

Call the event with data ***data*** (the instance of event definition class)

Example:

```
MyEvent.Call(new MyEvent(5));
```

```
void Call(params object[] args)
```

Call the event with event class constructor parameters

Example:

```
MyEvent.Call(5);
```

```
void Call()
```

Call the event with default data, also used if the event has no data.

Example:

```
MyEvent.Call();
```

Attributes

HandlerEvent

Applied to methods, marking them as event handlers.

Example:

```
[GlobalEvent.HandlerEvent]
void MyEventHandler(MyEvent ev)
{

}
```

ParameterEvent

Applied to events definition classes, defines the way of their processing.

Example:

```
[GlobalEvent.ParameterEvent(Requirement =
GlobalEvent.HandlerRequirementType.NonRequired)]
public class MyEvent : GlobalEvent.BaseEvent<MyEvent>
{
	public int EventData { get; private set; }

	public MyEvent(int data)
	{
		EventData = data;
	}
}
```

Variables:

Requirement – enumeration HandlerRequirementType, defining a way of processing of an event.

Enumerations

HandlerRequirementType

Required – indicates, that the event should have handler.

NonRequired – indicates, that handler not required.

LogicController

The main class which is responsible for initialization and execution of logics. Also carries out initialization of basic helpers for components code (global events, drag and drop, localization, etc.). In addition, executes all MonoBehaviour methods, which are present in components (Start, Update, FixedUpdate and LateUpdate are supported at the moment). This class is a singleton based on MonoBehaviour.

Variables:

Instance – the static instance of **LogicController** class.

Public methods:

```
void RunLogicExternal (LogicStorage logicStorage)
```

Initialize and execute logic for ***logicStorage*** storage. Upon completion, all *Start* methods which are present in the components of this logic will be called.

```
string RunLogicInstance(LogicStorage logicStorage, object data)
```

Execute and initialize the instantiated logic for ***logicStorage*** storage with ***data*** transferred to it. The ID of the running logic is returned.

```
void StopLogicInstance(string instanceId)
```

Clear the resources, allocated for an instantiated logic with the **instanceId** identifier.

LogicStorage

This class is a data storage for logic data.

Variables:

Id – identifier (GUID as string)

Name – logic name.

SceneName – the name of scene to which the logic belongs. It is used for recovery of logics.

Components – logic components data storage.

Links – the storage of data about the links between components.

Groups – the storage of data about the group components.

Static methods:

```
TextAsset Save (string path, LogicStorage storage)
```

Save logic as a text asset into the defined path.

```
LogicStorage Load (string path, string fileName)
```

Load logic from a file.

```
LogicStorage Load (TextAsset textAsset)
```

Load logic from a text asset.

Identifier

An abstract class for identification of an object by Id (GUID as string)

Variables:

Id – identifier (GUID string)

Component

Abstract class, the parent of all logic components. Derived from **ScriptableObject.**

Protected variables:

CoroutineHost – the scene object (MonoBehaviour), related to which coroutines are launched.

Public methods:

```
void Initialize(MonoBehaviour host)
```

Initialize the object for coroutine launching. Called on logic execution.

Virtual methods:

```
void Constructor()
```

This method is Awake analog, it's executed in the moment of component initialization at execution of logic.

Note: this method is the main place of components initialization, input points handlers e.t.c.

Variable<T>

Abstract Generic class for the definition of special subclass of logic components – variables. It's a base class for all variables, used in logic.

Variables:

Value – value, defined for the variable.

Events:

OnChanged – an event of variable value changing. It's called if the new value differs from the current value.

OnSet – an event of setting variable value. It's called always when variable gets a new value (even if it matches current value).

INPUT_POINT

Class, defining the component input point, which doesn't require data input.

Variables:

Handler – the handler of the input point event without external data.

INPUT_POINT<T>

Generic class, defining the component input point, which requires data input.

Variables:

Handler – the handler of the input point event with external data.

OUTPUT_POINT

Class, defining the component output point, which doesn't require data output.

Public methods:

```
void Execute()
```

Execute all input points handlers, which are connected with this output point.

OUTPUT_POINT<T>

Generic class, defining the component output point, which requires data output.

Public methods:

```
void Execute(T param)
```

Execute all input points handlers, which are connected with this output point with parameter ***param***.

VARIABLE_LINK<T>

Generic class, setting the reference to the logic variable of the type T in the component.

Variables:

Value – current value of the bound variable.

VariableWasSet – flag, showing that the reference to the logic variable was set.

Public methods:

```
void AddSetEventHandler(Variable<T>.Set handler)
```

Add the event handler for setting the value of bound logic variable.

```
void RemoveSetEventHandler(Variable<T>.Set handler)
```

Remove the event handler for setting the value of bound logic variable.

```
void AddChangedEventHandler(Variable<T>.Changed handler)
```

Add the event handler for changing the value of bound logic variable while setting it.

```
void RemoveChangedEventHandler(Variable<T>.Changed handler)
```

Remove the event handler for changing the value of bound logic variable while setting it.

Components

Abstractions

ArrayItemsAbstract

Abstract generic class, implementing the logic of working with array items. Used in the logic components development. Can return array item by index, and check the component existence by value.

ArraySamplingAbstract

Abstract generic class, implementing the logic of array items sampling. Used in the logic components development. Sampling is carried out in three modes: sequential, random and shuffle (random without repeats)

Variables:

Sampling – enumeration, defining the sampling type.

Loop – flag, indicating that sampling should be done in a cycle.

CastToTypeAbstract

Abstract Generic class, implementing the logic of type casting, used in the development of logic components.

Protected methods:

```
bool CastValue(T value)
```

Cast value ***value*** to the new type and return the result.

ForEachCycleAbstract

Abstract generic class, implementing cycled access to an array, used in the development of logic components.

GameObjectComponentAbstract

Abstract generic class, intended for the development of logic components, working with GameObject component.

Virtual methods:

```
void Start()
```

MonoBehaviour method for the component initialization.

```
void DoActionAfterValidation(Action action, Action failedValidationHandler = null)
```

Call an action with the component (with its preliminary validation).

GameObjectComponentSetGetAbstract

Abstract generic class, designed to develop logic components that implement the logic for setting and retrieving the value of the parameter of the GameObject component.

Variables:

HideFlag – flag specifying the set of input and output points of an component.

InternalValue – the internal value of the component to be set in the component parameters.

Abstract methods:

```
void ComponentGetValue(T value)
```

Set the value of the component parameter.

```
T ComponentSetValue ()
```

Get the value of the component parameter.

Virtual methods:

```
void Start()
```

MonoBehaviour method for the component initialization.

```
void DoActinoAfterValidation(Action action, Action failedValidationHandler = null)
```

Execute the action with a preliminary check for the existence of the component on the object.

```
void ValidateComponent ()
```

Validate the component (check for existence and obtain a component from the object).

IfConditionAbstract

Abstract Generic class, implementing the logic of comparing two values according to a given condition, used in the development of logic components.

Variables:

HideFlag - flag specifying the set of input and output points of an component.

Condition – condition for comparing two values.

ValueForCompare – Internal values for comparison.

Abstract methods:

```
bool Compare(T first, T second);
```

This method implements the direct logic of comparing two values depending on the ***Condition*** flag set.

IfEqualAbstract

Abstract generic class, which implements the logic of checking two values for equality, used in the development of logic components.

Virtual methods:

```
bool CompareEqual(T first, T second)
```

Check two values for equality with the method ***Equals***.

SendAbstract

Abstract generic class, which implements the logic of sending data to the component on demand, used in the development of logic components.

Variables:

Value – the value, sent on demand.

SetGetAbstract

Abstract generic class that implements the logic of setting and retrieving the value, used in the development of logic components.

Variables:

HideFlag – flag specifying the set of input and output points of an component.

InternalValue – the internal value of the component to be set.

Abstract methods:

```
void SetValue(T value)
```

Set value.

```
T GetValue()
```

Get value.

SetGetVariableAbstract

Abstract generic class that implements the logic of setting and retrieving the value of a logic variable, used in the development of logic components.

Variables:

HideFlag – flag specifying the set of input and output points of an component.

InternalValue – the internal value of the component to be set in the variable.

Abstract methods:

```
void SetValue(T value)
```
Set variable value.

```
T GetValue()
```

Get variable value.

SubscriberVariableAbstract

Abstract Generic class that implements the logic of subscribing to the events of setting and changing the value of logic variable, used in the development of logic components

SwitchAbstract

Abstract Generic class, that implements the logic of branching by the value, used in the development of logic components.

Variables:

SwitchValues – the list of values for switch logic.

Virtual methods:

```
bool CompareEqual(T first, T second)
```

Compare two values for equality.

```
string GetValueString(T value)
```

Return a string representation of a value. It is necessary to display the names of the component output points.

VLObjectAbstract

Abstract Generic class, intended for working with Unity objects, wrapped in **VSObject**.

Virtual methods:

```
void DoActionAfterValidation(Action action, Action errorHandler = null)
```

Call an action with the VSObject (with its preliminary validation).

Protected variables:

unityObject – the reference to the Unity object instance.

VLObjectSetAbstract

Abstract Generic class, implementing the logic of setting parameter value for Unity object, wrapped in **VSObject**.

Protected variables:

unityObject – the reference to the Unity object instance.

Abstract methods:

```
void SetValueToParam(SetParamType value)
```
Set parameter value.

```
SetParamType GetInternalValue()
```
Get the internal parameter value for setting.

VLObjectSetGetAbstract

Abstract Generic class, implementing the logic of setting and retrieving parameter value for Unity object, wrapped in **VSObject**.

Protected variables:

unityObject – the reference to the Unity object instance.

Variables:

HideFlag – flag specifying the set of input and output points of an component.

InternalValue – internal value of an component for setting into Unity object parameter.

Abstract methods:

```
void VLObjectSetValue (T value)
```

Set parameter value.

```
T VLObjectGetValue ()
```

Get parameter value.

Virtual methods:

```
void DoActionAfterValidation(Action action, Action failedValidationHandler = null)
```

Performing an action after object validation and calling the handler in case of validation failure.

Definitions

This section describes classes that are wrappers over various data necessary for the operation of components. All these classes have a unique PropertyDrawer for the convenience of their configuration in the inspector.

AnimationEventDefinition

Class for defining parameters of subscription to events of occurrence of certain frames in the animation.

Variables:

ClipIndex – the index of clip in the object animations list.

FrameEvents – the list of events by frame.

Classes

EventDescription

Class, describing the event for a frame.

Variables:

Name – the name of the event (identifier).

Frame – frame number.

LogicEventDefinition

Class, defining the event of data transfer from one logic to the other.

Variables:

ToId – identifier of the data receiver logic.

FromId – identifier of data sender logic.

LogicMessageDefinition

Class, defining the message for sending to all its subscribers.

Variables:

Id – message identifier.

InputAxisDefinition

Class, defining the configuration of input for Unity Input System with respect to the use of Axis commands.

Variables:

Name – Axis name.

MecanimParameterDefinition

A Class, defining the parameter, defined in Mechanim state machine tree.

Example of use:

```
public MecanimParameterDefinition Parameter = new
MecanimParameterDefinition(AnimatorControllerParameterType.Int);
```

Variables:

Name – parameter name.

DataType – parameter data type identifier.

Constructor:

```
MecanimParameterDefinition(AnimatorControllerParameterType dataType)
```

As an argument, an enumeration is passed that specifies the data type of the parameter in the animation controller.

PhysicEventsDefinition

The class that defines the events of the physical subsystem.

Public methods:

```
List<VSPhysics.PhysicsEventType> GetEvents()
```

Return the list of events for processing.

SceneDefinition

A class that defines a scene from the current Unity build settings.

Variables:

SceneName – name of the scene in the set.

SceneIndex – index of the scene in the set.

AssetBundleDefinition

A class that defines a asset bundle name.

Variables:

Name – asset bundle name.

Enumerations

SamplingType

Enumeration, defining sampling type.

Sequence – sequential sampling.

Random – random sampling.

Shuffle – random sampling without repeating values.

BranchingConditionType

Enumeration, defining branching conditions.

Equal – ***equal*** condition.

Less – ***less*** condition.

More – ***more*** condition.

MoreEqual – ***more or equal*** condition.

LessEqual – ***les or equal*** condition.

HideBranchingFlag

Enumeration specifying the set of points for components with branching by condition

None – all points are displayed.

HideInternal – points for comparison of external value with internal are hidden.

HideExternal – points for comparison of external values are hidden.

HideSetGetFlags

Enumeration defining a set of points for components of data setting and retrieving.

None – all points are displayed.

HideSet – setting points are hidden.

HideGet – getting point are hidden.

Others

Classes

AnimationEventHelper

A helper class that redirects animation events to the registered handlers, based on MonoBehaviour.

Public methods:

```
void RegisterEventHandler(Action<string> handler)
```

Register an external event handler for an animation event.

```
void UnregisterEventHandler(Action<string> handler)
```

Unregister an external event handler for an animation event.

MonoBehaviourSingleton

Abstract generic class is a singleton based on MonoBehaviour.

GameObjectPool

A class that implements the logic of the object pool manager. It is a singleton based on MonoBehaviour.

Static variables:

Instance – reference to an instance of the GameObjectPool class.

Public methods:

```
GameObject Create(GameObject prefab)
```

```
GameObject Create(GameObject prefab, Vector3 position, Quaternion rotation)
```

Create a clone of the ***prefab*** object. If the object is in the pool, it will be returned, otherwise a new object will be created via ***Instantiate***. The prefab name is used as the identifier of the object pool group.

```
void Destroy(string group, GameObject obj)
```

Delete an object. When you delete an object, it is not destroyed, but placed in the pool and deactivated.

```
void Clear(string group)
```

Clean the object pool group. Objects are removed from the game.

```
void Clear()
```

Full pool cleaning method. All objects in all groups are deleted. Cleaning is automatically performed when the scene is unloaded.

Randomizer

A class with helper methods for working with random numbers.

Public static methods:

```
int[] RandomIndices(int length)
```

Return a set of randomly rearranged indices in an array with length ***length.***

Serialization

A class with helpers methods implementing binary data serialization.

Public static methods:

```
void Serialize(object data, string path, string fileName)
```

Serialize data into a binary format and write the result to a file.

```
object Deserialize(string path, string fileName)
```

Deserialize binary data from a file.

```
object Deserialize(TextAsset textAsset)
```

Deserialize binary data from text asset.

Singleton

Abstract Generic class, implementing a classic singleton.

TweenHelper

A helper class containing methods that implement the logic for changing values over time.

Public static methods:

```
IEnumerator FromToEnumerator<T>(float time,
                                TweenLoopType loopType,
                                Func<float, T> tweenValue,
                                Func<T, bool> feedback,
                                Action complete = null)
```

Generic method for starting via StartCoroutine to change the value in time without peak.

time – the time for a change.

loopType – the type of a loop.

tweenValue – a delegate accepting time at the input and issuing the value corresponding to it.

feedback – a delegate accepting the current value and returning the continuation flag. If the flag is false, then the work of the coroutine is stopped.

Complete – this delegate is called after the time ***time*** elapsed, only in the mode without looping.

```
IEnumerator UpDownEnumarator<T>(float upTime,
                                float downTime,
                                TweenLoopType loopType,
                                Func<float, T> tweenUp,
                                Func<float, T> tweenDown,
                                Func<T, bool> feedback,
                                Action complete = null)
```

Generic method for starting via StartCoroutine to change the value in time with the peak.

upTime – the time to reach peak value.

downTime – the time to reach end value.

loopType – thy type of loop.

tweenUp – a delegate accepting the time at the input and giving out the value corresponding to it. Called when moving up to the peak value.

tweenDown – a delegate accepting the time at the input and giving out the value corresponding to it. Called when moving down to the end value.

feedback – a delegate accepting the current value and returning the continuation flag. If the flag is false, then the work of the coroutine is stopped.

Complete – this delegate is called after the time ***time*** elapsed, only in the mode without looping.

```
Vector2 Sinusoidal (Vector2 from, Vector2 to, float amplitude, float peaks, float t)
```

Return 2D position between ***from*** and ***to*** according to the time ***t***. The position is returned based on a sinusoidal function with amplitude ***amplitude*** and the number of peak ***peaks***.

```
Vector2 Cosinusoidal(Vector2 from, Vector2 to, float amplitude, float peaks, float t)
```

Return 2D position between ***from*** and ***to*** according to the time ***t***. The position is returned based on a cosinusoidal function with amplitude ***amplitude*** and the number of peaks ***peaks***.

```
Vector3 Sinusoidal(Vector3 from, Vector3 to, Vector3 axis, float amplitude, float peacks, float t)
```

Return 3D position between ***from*** and ***to*** according to the time ***t***. The position is returned relative to the axis ***axis*** based on a sinusoidal function with an amplitude ***amplitude*** and the number of peaks **peaks**.

```
Vector3 Cosinusoidal(Vector3 from, Vector3 to, Vector3 axis, float amplitude, float peacks, float t)
```

Return 3D position between ***from*** and ***to*** according to the time ***t***. The position is returned relative to the axis ***axis*** based on a cosinusoidal function with an amplitude ***amplitude*** and the number of peaks **peaks**.

VLObject

A wrapper class over Unity objects. Used in components to correctly save and restore links to prefabs and scene objects. Has a unique PropertyDrawer.

Constructor:

```
VLObject()
```

Default constructor.

```
VLObject(UnityEngine.Object obj)
```

Constructor, taking the reference to an Unity object.

Variables:

`HolderId` – identifier of object reference holder.

Id – identifier of object reference.

Properties:

Obj – Unity object reference.

Public methods:

```
T Get<T>() where T : UnityEngine.Object
```

A generic method that returns a reference to a Unity object with a type specification. The function returns null if casting is not possible.

Interfaces

IInputPointParse

The interface for obtaining a list of input points of an component. In the absence of implementation, the set of points is obtained through the reflection of the class.

Methods:

```
Dictionary<string, object> GetInputPoints()
```

Return a set of input points, as the name of a point and a reference to the class corresponding to it.

IOutputPointParse

The interface for retrieving a list of component output points. In the absence of implementation, the set of points is obtained through the reflection of the class.

Methods:

```
Dictionary<string, object> GetOutputPoints()
```

Return a set of output points, as the name of the point and a reference to the class corresponding to it.

IPhysicEventHandler

The interface for subscribing and unsubscribing to events of the Unity physical subsystem.

Methods:

```
void RegisterHandler<T>(Action<T> handler)
```

Generic method for registering.

```
void UnregisterHandler<T>(Action<T> handler)
```

Generic method for unregistering.

IWrapperData

The interface used to convert the data needed to display information in the component parameters section in the logic editor. In the absence of implementation, the data is obtained through the reflection of class fields.

Methods:

```
KeyValuePair<string, string> GetInfo()
```

Return a pair of values: the name of parameter and a string representation of the parameter value.

Attributes

ComponentDefinitionAttribute

Description attribute of the logic component.

Parameters:

Name – the name in the catalog

Path – the folder in the catalog.

Tooltip – the description in the catalog.

Color – the color of header and the color in the catalog

ExecuteOrderAttribue

An attribute specifying the order of execution of the MonoBehaviour methods (Start, Update, etc.). The methods are executed in ascending order. The first method is the one with the lowest value. Methods without an attribute are executed after the methods with it.

Parameters:

Order – ordinal number of execution.

HideInEditorAttribute

An attribute used to hide the reference to a variable in an component in the logic editor. Applies only to VARIABLE_LINK. By default, all links are displayed in the editor.

ViewInEditorAttribute

The attribute used to display the open variable in the component's parameters section in the logic editor. By default, all open variables are hidden.

ViewInDebugModeAttribute

The attribute used to display the field value in debug mode.

TypeConstraintAttribute

The attribute used to restrict a type to a VLObject.

Enumerations

TweenLoopType

Enumeration defining the loop type when changing values.

Once – no loop mode.

Loop – cycle loop mode.

PingPong – ping-pong loop mode.

TweenMotionType

Enumeration defining the way to change the value for vectors. It is used in the component inspector.

Linear – linear change.

Sin – sinusoidal change.

Cos – cosinusoidal change.

Visual Logic Editor

VLEDebug

A specialized class used in Unity editor mode for debugging logic.

Public properties:

IsBreak – flag indicating that the logic is stopped

Events:

OnTracertStackChanged –events triggered by changing data for link tracing in logics.

OnNextStep – events triggered for moving to the next component of the logic when step-by-step debugging.

Public static methods:

```
void TRACERT(string logic, string link)
```

```
void TRACERT(string logic, string link, string data)
```

Call the link tracing in the logic.

```
void Break()
```

Stop the logic execution.

```
void Next()
```

Move to the next logic component when step-by-step debugging

```
void Play()
```

Stop the debugging and continue logic execution in normal mode.

```
void Clear()
```

Clean the debug data.

Classes

TracertData

A class that contains the data for link tracing in the logic when debugging.

Properties:

Logic – logic identifier.

Link – link identifier.

Data – transmitted data.

VLEditor

A namespace that contains classes, interfaces, and so on, implementing the logic of the logic editor.

Classes

InputOutputPointData

A class describing the input and output points of an component

Variables:

Name – point name.

Tooltip – a hint, the default is name of the point, otherwise the data from the Tooltip attribute.

DataType – point data type

PointType – point type (input or output).

LinkingPointData

A class describing the parameters for linking the points of components.

Variables:

Component – reference to an component containing a point.

PointName – the name of a point for connection.

DataType – point data type.

Center – point position.

VLEWidget

Abstract base class for displaying components in the logic editor.

Public properties:

Id – component identifier.

Name – component name.

Comment – component comment.

Group – component group.

Description – the description of component type.

WidgetColor – the color of component header.

Instance – the reference to component type instance.

InstanceType – component type.

IsSelected – the flag of component selection in a logic.

Area – component drawing area.

Protected properties:

isMoved – component dragging flag.

componentStorage – reference to an instance of the data storage class of component.

Protected variables:

mainWidget – main component widget.

headerWidget – component header widget.

descriptionWidget – component type description widget.

bodyWidget – component body widget.

Constructor:

```
VLEWidget(Core.LogicStorage.ComponentsStorage.ComponentData componentStorageData)
```

Accepts as a parameter an instance of the data storage class of the component.

Public methods:

```
void Select(bool state)
```

Select and deselect the component.

```
void SetPosition(Vector2 newPosition)
```

Set the new position for the component.

```
void Draw(bool lockMouse)
```

Draw an component. The lockMouse flag displays the cursor lock. When the cursor is locked, all events from the user are ignored.

Virtual methods:

```
void Dispose()
```

Clear the memory occupied by the component, used if the component loads unmanaged resources.

```
void HandleEvent(Event ev)
```

Handle user events.

```
void MouseDown(Event ev)
```

Handle mouse button pressing event.

```
void MouseDrag(Event ev)
```

Handle mouse drag event.

`void MouseUp(Event ev)`

Handle mouse button releasing event.

`void HeaderDoubleClick() { }`

Process double-clicking on the widget header.

Classes

WidgetComponentsContainer

An abstract class that implements the logic of the components widget container.

Public virtual properties:

Area – container drawing area.

Position – container local position.

WorldPosition – container world position.

Protected properties:

widgets – container widgets list.

Protected variables:

widgetRect – current component drawing area.

rootContainer – container root widget.

Constructor:

`WidgetComponentContainer(IWidgetContainer root)`

Accepts as a parameter a reference to the root container.

Virtual methods:

`void RegisterStyle()`

Register GUIStyle, used to draw the widget.

`void Draw()`

Draw a container.

`void AddWidget(IChildWidget widget)`

Add a widget to a container.

`void RemoveWidget(IChildWidget widget)`

Delete a widget from a container.

`void ClearWidget()`

Clean the container from all widgets.

`void ResizeWidgetContainer()`

Update container size.

Protected methods:

```
void UpdateRootContainer()
```

Update the root container in case it exists.

ChildWidget

An abstract class that implements the logic of the component widget, contained in the container.

Public virtual properties:

Area – widget drawing area.

Position – widget local position.

WorldPosition – widget world position.

Protected variables:

widgetRect – current component drawing area.

Virtual methods:

```
void RegisterStyle()
```

Register GUIStyle, used to draw the widget.

Abstract methods:

```
void Draw()
```

Draw a widget.

MainWidget

A class that implements the logic of the root container of an component widget.

BodyWidget

A class that implements the logic of the component body, it is a container for other widgets.

HeaderWidget

A class that implements the logic of an component header widget.

DescriptionWidget

A class that implements the logic of the component type description widget.

VLEComponent

A class that implements the logic of drawing a logic component, except for variables. Inherited from VLEWidget.

Public properties:

IsMinimized – component minimization flag.

IsHideParameter – flag for hiding a region with component parameters.

IsInversion – component points inversion flag.

IsCanMinimize – flag indicating that the component can be minimized.

TracertState – flag indicating the trace state of the component when debugging

TracertInProgress – flag indicating that the component is in trace mode.

Protected variables:

pointsContainer – container of widgets.

pointsCollapsedContainer – container of widgets in minimized state.

parametersContainer – the root container of component parameters area widgets.

parameterButtonWidget – widget of component parameters button.

parameterValuesContainter – container of widgets with a description of each component parameter.

Public methods:

```
void MakeTracert(bool state, bool inProgress)
```

Trace an component.

```
void SetCollapseComponent(bool state)
```

Minimize an component.

```
void SetCollapseParameter(bool state)
```

Hide component parameters.

```
void SetInversionPoints(bool state)
```

Invert the component points.

```
Rect GetInputPointRect(string pointName)
```

Return the input point drawing region by a name.

```
Rect GetOutputPointRect(string pointName)
```

Return the output point drawing region by a name.

```
int InputPointCanBeAccepted(Vector2 position, Type type)
```

Check the possibility of establishing a connection with a point. The check is carried out by the cursor position and data type. The result of the check is returned as: 1 - connection is possible, -1 - connection impossible due to non-correspondence of types, 0 - connection is not possible by position.

```
LinkingPointData GetLinkAcceptedPoint()
```

Return the data for establishing a connection with a point that returned 1 in the connection possibility check method ***InputPointCanBeAccepted***.

Public virtual methods:

```
void UpdateParameterValues()
```

Method of forced updating of component parameters values.

Protected virtual methods:

```
void PrepareChildWidget()
```

Prepare component child widgets.

Protected methods:

```
LinkingPointData GetDraggedOutputPoint(Vector2 position)
```

Return data for connection the component point with a check at the cursor position, if the cursor is not on the point, null is returned. Only the output points are tested.

```
void UpdateInputPointsChildWidget()
```

Update the input points widgets.

```
void UpdateOutputPointsChildWidget()
```

Update the output points widgets.

```
InputOutputPointData GetPointData(string fieldName, object pointData,
PointTypeEnum pointType)
```

Return the point information from the instance of the component type class. Used when updating point widgets.

Classes

PointsContainer

A class that implements the logic of point widgets container.

PointWidget

A class that implements the logic of the component point widget.

ParametersContainer

A class that implements the logic of the widget container of an component parameter area.

ParameterValuesContainer

A class that implements the logic of the widget container of an component parameters description.

ParameterButtonWidget

A class that implements the logic of the component's hide / show parameters button.

ParameterValueWidget

A class that implements the logic of the widget of component parameter description.

VariableLinkWidget

Class that implements the logic of the widget of the reference to the logic variable.

MenuWidget

A class that implements the logic of the component menu button widget.

VLEVariable

A class that implements the logic of drawing of logic variable. Inherited from VLEWidget.

Protected variables:

valueData – data for displaying the value of a variable.

valueWidget – widget for displaying the value of a variable.

Protected virtual methods:

```
void PrepareChildWidget()
```

Prepare component widgets.

Public virtual methods:

```
void UpdateParameterValues()
```

Method for forced updating of variable values.

Classes

VariableValueWidget

A class that implements the logic of variable value widget.

VLELink

A class that implements the logic of drawing links between components.

Public properties:

Id – link identifier.

SourceComponent – a reference to the source component widget.

TargetComponent – a reference to the target component widget.

OutputPoint – output point name.

InputPoint – input point name.

IsSelected – a flag, indicating that the link is selected.

IsCorrupted – a flag, indicating that the link is corrupted.

LinkColor – the color of link.

CallOrder – the order of execution of the link.

Constructor:

```
VSELink(LogicStorage.LinksStorage.LinkData linkStorage, IComponentWidget source,
IComponentWidget target)
```

As parameters, it takes references to the link data storage class and references to the connected components of the logic.

Public methods:

```
void Draw(bool lockMouse)
```

Link drawing function. The ***lockMouse*** flag indicates locking the cursor, in which case events from the user are ignored (you cannot select a link, etc.)

```
bool Contains(Vector2 position)
```

The function of checking the cursor position relative to the link, returns true if the cursor is in the link area.

```
void MakeTracert(bool state, string data)
```

Trace the link.

```
void Select(bool state)
```

Select the link in the logic editor.

VLEGroup

A class for displaying a group of widgets

Public properties:

Id – group identifier.

Bounds – the bounds of the group drawing rectangle (calculated automatically)

IsMoving – a flag indicating that the group is in the drag mode (Drag and Drop)

Widgets – list of widgets in a group

Constructor:

```
VLEGroup(LogicStorage.GroupsStorage.GroupData groupData)
```

Takes a link to the data from the logic storage with the group description as a parameter.

Public Methods:

```
void Draw(bool lockMouse)
```

Draw a group using the mouse cursor lock flag.

```
void AddComponent(ILogicWidget widget)
```

Add a widget to a group.

```
bool RemoveComponent(ILogicWidget widget)
```

Remove a widget from a group. Returns true if no widgets are left in the group.

VLECommon

A class containing static utility methods for working with selected objects and links, and also used to transfer the current states of the logic editor to all subsystems.

События:

OnSelectComponentChanged – event of changing the currently selected component (selecting a new component)

OnSelectedLogicChanged – event of changing the currently loaded logic (loading a new logic)

OnSelectedLogicUpdated – event of updating the state of the current logic.

OnComponentSizeChanged – event of changing the size component.

Static variables:

SelectedWidgets – list of currently selected component widgets.

SelectedLinks – list of currently selected links between components.

UnsavedChanges – counter of unsaved changes in the logic.

BufferCounter – counter of components in the buffer.

SceneLogicController – reference to the instance of the current scene logic controller.

Public static properties:

CurrentLogic – the reference to the current logic storage.

SelectedWidget – the currently selected component widget (null if several widgets are selected).

SelectedLink – the currently selected link between components (null if multiple links are selected).

IsMultipleComponents – flag indicating that several components are selected.

IsMultipleLinks – flag indicating that several links are selected.

PantheaAssembly – the reference to an assembly containing class types describing the logic of all components.

RuntimeEditorAssembly – the reference to an assembly containing class types describing custom components widgets.

Setting – the reference to an instance of the class that contains global settings for the logic editor.

IndexOfCurentLogic – the index of the current open logic in the list of scene logics.

Public static methods:

`void` UpdateLogic()

Call the update events for the current logic.

```
void ComponentChangedSize(IComponentWidget component, float deltaChange)
```

Call the changed size events for the component.

```
void ClearSelection()
```

Clear all selection states (controller, logic, lists of components and links). Called when the editor working is finished.

```
void ClearSelectedComponents()
```

Clear the list of selected components.

```
void ClearSelectedLinks()
```

Clear the list of selected links.

```
void ClearSetting()
```

Clear the settings.

```
void ClearAssembly()
```

Clear the assemblies.

```
void ResetUnsavedChanges()
```

Reset unsaved changes counter.

VLETime

Wrapper class for obtaining system time in the logic editor.

Public static properties:

Time – the current time since the start of the Unity editor.

DeltaTime – the time elapsed since the last update of the logic editor.

Public static methods:

`void Start()`

Initialize the class.

`void Update()`

Update the class state.

VLEditorUtils

A class containing a set of utility methods necessary for the operation of the logic editor

VLEExtension

A static class that contains wrapper classes over the data structures needed to draw components, considering the size of the virtual pixel in screen pixels (needed to change the scale of the logic).

VLEVirtualRect

A wrapper class above the Rect structure, which is its virtual copy and serves to translate the original value into a new value considering the virtual pixel size in screen pixels.

VLEVirtualPosition

A wrapper class over the Vector2 structure, which is its virtual copy and serves to translate the original value into a new value considering the virtual pixel size in screen pixels.

VLEVirualFontSize

A wrapper class above the font size (integer value), which is its virtual copy and serves to translate the original value into a new value considering the virtual pixel size in screen pixels.

Interfaces

ILogicWidget

Base interface describing the widget of an component in the logic editor.

Properties:

Id – component identifier from the storage.

Name – component name in the logic.

Comment – component comment in the logic.

Description – component class descriptions.

Instance – a reference to the class instance.

InstanceType – component class type.

WidgetColor – component widget header color.

Area – current drawing area of the component widget.

IsSelected – component widget selection flag.

Methods:

`void Draw(bool lockMouse)`
Draw an component widget considering the mouse cursor lock flag value.

`void SetPosition(Vector2 newPosition)`
Set the position of the component widget.

`void Select(bool state)`

Select the component widget.

IComponentWidget

The interface describing the widget of the components except for a variable (all components that implement logic).

Properties:

IsMinimized – component minimization state flag.

IsHideParameter – flag indicating the state of component parameters hiding.

IsInversion – component points inversion flag.

IsCanMinimize - flag indicating that component can be minimized.

TracertState – state flag for trace component in debug mode

TracertInProgress – trace progress state flag.

Methods:

```
Rect GetInputPointRect(string pointName)
```

Get the drawing area of the input point widget by the point name. Used to draw links between components.

```
Rect GetOutputPointRect(string pointName)
```

Get the drawing area of the output point widget by the point name. Used to draw links between components.

```
int InputPointCanBeAccepted(Vector2 position, Type type)
```

Check the possibility of establishing the link between the output point with the type of data ***type*** and the input point of the component. The check is performed at the mouse cursor position, if the positions do not match, the value 0 is returned, then the type match is checked, if the types do not match, the value -1 is returned, otherwise 1. The method is used to draw the established link.

```
LinkingPointData GetLinkAcceptedPoint()
```

Return the data of the point with which you can make a link, the data is the result of a check from the ***InputPointCanBeAccepted*** method, if it returns a value of 1, otherwise null is returned.

```
void SetCollapseComponent(bool state)
```

Setting the component widget minimization state.

```
void SetCollapseParameter(bool state)
```

Set the state of parameters hiding in the component widget.

```
void SetInversionPoints(bool state)
```

Set the state of points inversion in the component widget.

```
void MakeTracert(bool state, bool inProgress)
```

Trace the component in the debug mode.

ILink

An interface describing the link between components.

Properties:

Id – link identifier from the storage.

SourceComponent – the reference to the link source component widget.

TargetComponent – the reference to the link target component widget

OutputPoint – the name of output point of source component.

InputPoint – the name of output point of target component.

IsSelected – a flag, indicating that the link is selected.

IsCorrupted – a flag, indicating that the link is corrupted.

Methods:

```
void Draw(bool lockMouse)
```
Draw the link considering the mouse cursor lock flag.

```
void Select(bool state)
```
Select the link.

```
bool Contains(Vector2 position)
```
Check the position of the mouse pointer relative to the link.

```
void MakeTracert(bool state, string data)
```

Trace the link in debug mode with the display of data transmitted over it.

IWidgetsGroup

An interface used to group widgets in a logic.

Properties:

Id – group identifier.

Bounds – the bounds of the group drawing rectangle (calculated automatically).

IsMoving – a flag indicating that the group is in the drag mode (Drag and Drop)

Widgets – a list of widgets in a group.

Methods:

```
void Draw(bool lockMouse)
```
Draw a group using the mouse cursor lock flag.

```
void AddComponent(ILogicWidget widget)
```

Add a widget to a group.

```
bool RemoveComponent(ILogicWidget widget)
```

Remove a widget from a group. Returns true if no widgets are left in the group.

IParameterContainer

The interface used for the components that contain parameters to display in the widget.

Methods:

`void UpdateParameterValues()`

Method for forced updating the values of parameters, displayed in the widget.

IWidgetContainer

An interface used for internal widgets of an component that are containers for other widgets.

Methods:

`void AddWidget(IChildWidget widget)`

Add a widget to a container.

`void RemoveWidget(IChildWidget widget)`

Remove a widget from a container.

`void ClearWidget()`

Clear the container from all widgets.

`void ResizeWidgetContainer()`

Container widget size update function.

IChildWidget

The interface for describing the internal widgets of an component.

Properties:

Area – widget drawing area.

Position – widget local position.

WorldPosition – widget world position.

Methods:

`void RegisterStyle()`

Register GUIStyle used to draw the widget.

`void Draw()`

Draw the widget.

Attributes

VLECustomComponentDrawerAttribute

The attribute used for the classes customizing a widget of a component displayed in the editor.

Parameters:

ComponentType – type of the class of the component whose widget is customized.

Enumerations

PointTypeEnum

Enumeration with a description of component point types.

Input – input point.

Output – output point.

Others

DefinitionPropertyDrawerAbstract

An abstract class that is base for custom property editors that describe definitions. Used to ensure that all definitions are displayed in the inspector in a single style.

Abstract methods:

```
void Draw(SerializedProperty property)
```

Draw Properties values.