Last edition 28.11.2017

API

Introduction

Welcome to **Panthea VS** programming guide. This document contains the description of all the classes, structs and enumerations, necessary for the development of elements and their visual representation in the diagram editor.

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

Visual Script Core

VSCore

This namespace contains the classes, implementing the core of Panthea VS system, responsible for diagrams 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()`

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.

DiagramController

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

Variables:

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

Public methods:

```
void RunDiagramExternal (DiagramStorage diagramStorage)
```

Initialize and execute diagram for ***diagramStorage*** storage. Upon completion, all *Start* methods which are present in the elements of this diagram will be called.

```
string RunDiagramInstance(DiagramStorage diagramStorage, object data)
```

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

```
void StopDiagramInstance(string instanceId)
```

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

DiagramStorage

This class is a data storage for diagram data.

Variables:

Id – identifier (GUID as string)

Name – diagram name.

SceneName – the name of scene to which the diagram belongs. It is used for recovery of diagrams.

Elements – diagram elements data storage.

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

Static methods:

```
TextAsset SaveDiagram(string path, DiagramStorage storage)
```

Save diagram as a text asset into the defined path.

```
DiagramStorage LoadDiagram(string path, string fileName)
```

Load diagram from a file.

```
DiagramStorage LoadDiagram(TextAsset textAsset)
```

Load diagram from a text asset.

```
string GetAssetPath(string fileName)
```

Get the path to asset relative to Assets folder.

Identifier

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

Variables:

Id – identifier (GUID string)

Element

Abstract class, the parent of all diagram elements. 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 diagram execution.

Virtual methods:

```
void Constructor()
```

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

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

Variable<T>

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

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 element 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 element input point, which requires data input.

Variables:

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

OUTPUT_POINT

Class, defining the element 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 element 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 diagram variable of the type T in the element.

Variables:

Value – current value of the bound variable.

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

Public methods:

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

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

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

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

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

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

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

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

Elements

Abstractions

ArrayItemsAbstract

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

ArraySamplingAbstract

Abstract generic class, implementing the logic of array items sampling. Used in the diagram elements 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 diagram elements.

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 diagram elements.

GameObjectComponentAbstract

Abstract Generic class, intended for the development of diagram elements, working with GameObject component.

Protected methods:

```
void ValidateComponent()
```

Validate component (check for existence and obtaining a component from object)

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 diagram elements 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 element.

InternalValue – the internal value of the element 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 diagram elements.

Variables:

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

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 diagram elements.

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 element on demand, used in the development of diagram elements.

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 diagram elements.

Variables:

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

InternalValue – the internal value of the element 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 diagram variable, used in the development of diagram elements.

Variables:

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

InternalValue – the internal value of the element 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 diagram variable, used in the development of diagram elements

SwitchAbstract

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

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 element output points.

TweenUpDownAbstract

Abstract Generic class, that implements the logic of changing a value with a peak, used in the development of diagram elements.

Variables:

LoopType – flag, setting the way to loop the value change.

UpTime – peak time value.

DownTime – end time value.

ResetToFromIfBreak – flag indicating that if the operation of the logic is interrupted, the changed value will be reset to the initial value.

DefaultFromValue – default start value.

DefaultUpValue – default peak value.

DefaultDownValue – default end value.

Abstract methods:

```
T Tween(T from, T to, float t)
```

Return value between ***from*** and ***to*** depending on time value ***t***, where ***t*** is normalized value between 0f and 1f.

TweenFromToAbstract

Abstract Generic class, that implements the logic of value changing, used in the development of diagram elements.

Variables:

LoopType – flag, setting the way to loop the value change.

Time – the time for the end value reaching.

ResetToFromIfBreak – – flag indicating that if the operation of the logic is interrupted, the changed value will be reset to the initial value.

DefaultFromValue – default start value.

DefaultToValue – default end value.

Abstract methods:

```
T Tween(T from, T to, float t)
```

Return value between ***from*** and ***to*** depending on time value ***t***, where ***t*** is normalized value between 0f and 1f.

VSObjectAbstract

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.

VSObjectSetAbstract

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.

VSObjectSetGetAbstract

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 element.

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

Abstract methods:

```
void VSObjectSetValue (T value)
```

Set parameter value.

```
T VSObjectGetValue ()
```

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 elements. 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.

DiagramEventDefinition

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

Variables:

ToId – identifier of the data receiver diagram.

FromId – identifier of data sender diagram.

DiagramMessageDefinition

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:

SceneIndex – Index of the scene in the set.

SoundDefinition

A class that defines the parameters of a single sound

Variables:

Name – sound name.

Channel – the name of the channel in which the sound should be played.

SoundGroupDefinition

A class that defines the parameters of a group of sounds

Variables:

Group – name of a group of sounds.

TweenMotionDefinition

A class that defines the motion between two points of space.

Variables:

Motion – the type of motion.

Amplitude – Amplitude (applies if the motion is sinusoidal or cosine).

Peaks – number of sine or cosine peaks.

AxisFluctuation – specify an axis along which there is a fluctuation of the position along the sinusoid or cosine wave.

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 elements 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 elements of data setting and retrieving.

None – all points are displayed.

HideSet – setting points are hidden.

HideGet – getting point are hidden.

DragDrop

Namespace that contains classes, implementing the logic for capturing, dragging and dropping an object.

IDropTarget

Interface, defining the object drop target.

Methods:

`bool DropAreaContainsPoint(Vector3 point)`
Check if a point is in the target area for a drop.

`bool CollectThisObject(string id)`
Verify that the target for the drop placed an object with an identifier ***id*** in its collection.

`void DropSuccessful(string id, GameObject gameObject)`
The event for successful dropping of the object ***gameObject*** with the identifier ***id*** to the target.

`void DropFailed()`

The event for failed dropping of the object.

DragDropHelper

A helper class that implements working with drop targets for a dragged object. This class is a singleton based on MonoBehaviour.

Static variables:

Instance – the instance of DragDropHelper class.

Public methods:

`void RegisterDropTarget(IDropTarget dropTarget)`

Register drop target.

`void UnregisterDropTarget(IDropTarget dropTarget)`

Unregister drop target.

`bool DropDraggedObject(string id, GameObject draggedObject)`

Check the result of dropping the object on target.

DragSourceHelper

A helper class implementing the logic of object dragging. Implements following Unity interfaces: IBeginDragHandler, IEndDragHandler, IDragHandler. Class is based on MonoBehaviour.

Public methods:

`void RegisterBeginDragHanlder(Action<PointerEventData> handler)`

Register external handler for the event of the beginning of object dragging.

`void RegisterEndDragHanlder(Action<PointerEventData> handler)`

Register external handler for the event of the end of object dragging.

`void RegisterDragHanlder(Action<PointerEventData> handler)`

Register external handler for the event of the object dragging.

Localization

A namespace that contains classes implementing the logic of the game localization system.

GameLocalization

A helper class that implements the reception and processing of events in the game localization system. This class is a singleton based on MonoBehaviour.

Public methods:

`void SubscribeTag(string tag, Action<object> handler)`

Subscribe to a data change event corresponding to the localization tag.

`void UnsubscribeTag(string tag, Action<object> handler)`

Unsubscribe from a data change event corresponding to the localization tag.

`void SetLanguage(LanguageData language)`

Set the current language for the game.

`void SetLocalizationTags(List<LocalizationTagParameter> tags)`

Set the list of localization tags.

Classes

LanguageChanged

A class that describes the global event of changing the current language of the game.

LanguageData

A class that describes the localization data for the language used in the game.

Variables:

Name – displayed name of the language, also used as an identifier.

Resources – list of localization resources corresponding to the language.

LocalizationResource

The wrapper class over localization resource data.

Variables:

StringData – a text string.

FontData – font for the text

SpriteData – sprite

TextureData - texture

AudioData – audio clip.

LocalizationData

This class is a storage of all localization data. It is inherited from ScritableObject. It is stored in the game as an asset that can be downloaded from the outside.

Variables:

Languages – a list of localization resources data by languages used in the game

Tags – list of localization tag information

LocalizationTagParameter

Class describing the parameters of the localization tag

Variables:

Name – tag name (identifier)

ResourceType – the type of localization resource data for the corresponding tag. Sets the tag filter by data type.

LocalizationResourceType

Enumeration defining the data type of the localization resource.

Text – resource is text.

Image – resource is sprite image.

Texture – resource is texture.

Audio – resource is audio clip.

LocalizationTagDefinition

The class that defines the tag used for localization, used in diagram elements. Has a unique PropertyDrawer for easy configuration.

Variables:

Tag – tag name.

ResourceType – resource data type.

Public methods:

```
void SubsribeToChanged(Action<object> handler)
```

Subscribe to the event of changing localization resource data, corresponding to tag.

```
void Unsubscribe()
```

Unsubscribe from localization resource data change event.

LocalizedResourceDefinition

A class that describes the localized resource in the game.

Constructor:

```
LocalizedResourceDefinition(LocalizationResourceType resourceType)
```

As a parameter, it takes the data type of the resource that is used to filter the tags.

Variables:

IsUsed – flag that determines the use of localization for the resource of the game.

Tag – localization tag name.

Public methods:

```
void SubsribeToChanged(Action<object> handler)
```

Subscribe to the event of changing localization resource data, corresponding to tag.

```
void Unsubscribe()
```

Unsubscribe from localization resource data change event.

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.

PoolObjectManager

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 PoolObjectManagerr 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.

SoundHelper

A helper class that contains the methods for working with sounds in a scene.

Public static methods:

```
Dictionary<string, Dictionary<string, AudioSource>> FindAudioSources()
```

Return all active sounds of the scene in the form of a set grouped by channels. The name of the channel is the name of the MixerGroup, in case it is not set, the sound will be placed in the Default channel.

SoundEngine

A class is superstructure over the Unity sound subsystem, it is a singleton based on MonoBehaviour.

Public methods:

```
void PlaySingleSound(string name, string channel = "Default", SoundChannelMode
channelMode = SoundChannelMode.Default, float delay = 0f, Action feedback = null)
```

Play the sound with the name ***name*** in the channel ***channel***, in the ***channelMode*** mode, with a delay **delay**. After the end of the sound, feedback is called.

```
void StopSound(string name, string channel = "Default")
```

Stop the sound with the name ***name*** in the channel ***channel.***

```
void PauseSound(string name, string channel = "Default")
```

Pause the sound with the name ***name*** in the channel ***channel.***

```
void PlayGroupSound(string group, SoundGroupPlayMode playMode =
SoundGroupPlayMode.Random, float delay = 0f, Action feedback = null)
```

Play a group of sounds in the ***playMode*** mode with a delay ***delay***. After the sound is over, feedback is called. The group is determined by the MixingGroup parameter.

```
void StopSound(string group = "Default")
```

Stop the sound, started from a group.

```
void PauseSound(string group = "Default")
```

Pause the sound, started from a group.

```
float GetSoundLength(string name, string channel = "Default")
```

Return the length of a sound named ***name*** from the channel ***channel*** in seconds.

```
float GetSoundVolume(string name, string channel = "Default")
```

Return the current volume of the sound with the name ***name*** from the channel ***channel.***

```
float GetSoundVolume(string group = "Default")
```

Return the current volume of the sound, started from a group.

```
float GetSoundPitch(string name, string channel = "Default")
```

Return the current pitch of the sound with the name ***name*** from the channel ***channel.***

```
float GetSoundPitch(string group = "Default")
```

Return the current pitch of the sound, started from a group.

```
void SetSoundVolume(float volume, string name, string channel = "Default")
```

Set the volume to the sound ***name*** from the channel ***channel.***

```
void SetSoundVolume(float volume, string group = "Default")
```

Set the volume to the sound in a group.

```
void SetSoundPitch(float pitch, string name, string channel = "Default")
```

Set the pitch to the sound ***name*** from the channel ***channel***.

```
void SetSoundPitch(float pitch, string group = "Default")
```

Set the pitch to the sound in a group.

```
bool SoundIsPlaying(string name, string channel = "Default")
```

Return the current state of the sound with the name ***name*** from the channel ***channel.***

```
float GetSoundTime(string name, string channel = "Default")
```

Return the current time position of the sound with the name ***name*** from the channel ***channel.***

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**.

VSCollision

This section describes the helper classes for working with events of physical collision of objects.

VSCollisionEventHandler

Generic class that implements the logic of subscription and unsubscription to events of physical collision of objects.

Protected variables:

Subscribers – the list of event subscribers.

Public methods:

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

Generic method for registration of event handler.

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

Generic method for unregistration of event handler.

VSCollisionEnterHandler

Helper class for redirecting the event of the beginning of a physical collision of 3D objects.

VSCollisionExitHandler

Helper class for redirecting the event of the end of a physical collision of 3D objects.

VSCollisionStayHandler

Helper class for redirecting the event of a physical collision of 3D objects.

VSCollisionEnter2DHandler

Helper class for redirecting the event of the beginning of a physical collision of 2D objects.

VSCollisionExit2DHandler

Helper class for redirecting the event of the end of a physical collision of 2D objects.

VSCollisionStay2DHandler

Helper class for redirecting the event of a physical collision of 2D objects.

VSTrigger

This section contains a description of the helper classes for working with the events of the physical collision of an object and a trigger.

VSTriggerEventHandler

Generic class that implements the logic of the subscription and unsubscription for events of physical collision of objects.

Protected variables:

Subscribers – the list of event subscribers.

Public methods:

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

Generic method for registration of event handler.

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

Generic method for unregistration of event handler.

VSTriggerEnterHandler

Helper class for redirecting the event of the beginning of a physical collision of 3D objects.

VSTriggerExitHandler

Helper class for redirecting the event of the end of a physical collision of 3D objects.

VSTriggerStayHandler

Helper class for redirecting the event of a physical collision of 3D objects.

VSTriggerEnter2DHandler

Helper class for redirecting the event of the beginning of a physical collision of 2D objects.

VSTriggerExit2DHandler

Helper class for redirecting the event of the end of a physical collision of 2D objects.

VSTriggerStay2DHandler

Helper class for redirecting the event of a physical collision of 2D objects.

VSPhysics

A helper class containing the functions of subscribing and unsubscribing to events of the physical subsystem.

Public static methods:

```
void RegisterHandler(UnityEngine.Object target, PhysicsEventType eventType,
Action<Collider> handler)
```

Register the event handler of 3D physical collisions of the object and the trigger ***target***.

```
void RegisterHandler(UnityEngine.Object target, PhysicsEventType eventType,
Action<Collision> handler)
```

Register the event handler of 3D physical collisions of the object ***target***.

```
void RegisterHandler(UnityEngine.Object target, PhysicsEventType eventType,
Action<Collider2D> handler)
```

Register the event handler of 2D physical collisions of the object and the trigger ***target***.

```
void RegisterHandler(UnityEngine.Object target, PhysicsEventType eventType,
Action<Collision2D> handler)
```

Register the event handler of 3D physical collisions of the object ***target***.

```
void UnregisterHandler(UnityEngine.Object target, PhysicsEventType eventType,
Action<Collider> handler)
```

Unregister the event handler of 3D physical collisions of the object and the trigger ***target***.

```
void UnregisterHandler(UnityEngine.Object target, PhysicsEventType eventType,
Action<Collision> handler)
```

Unregister the event handler of 3D physical collisions of the object ***target***.

```
void UnregisterHandler(UnityEngine.Object target, PhysicsEventType eventType,
Action<Collider2D> handler)
```

Unregister the event handler of 2D physical collisions of the object and the trigger ***target***.

```
void UnregisterHandler(UnityEngine.Object target, PhysicsEventType eventType,
Action<Collision2D> handler)
```

Unregister the event handler of 2D physical collisions of the object ***target***.

Enumerations:

PhysicsEventType – an enumeration describing the types of physical simulation events.

VSObject

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

Constructor:

```
VSObject()
```

Default constructor.

```
VSObject(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.

Structures

VSEColor

The wrapper structure for Color, used to save colors when using binary serialization. Contains a set of color constants in the Html format used to pass to attribute parameters.

Interfaces

IInputPointParse

The interface for obtaining a list of input points of an element. 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 element 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 element parameters section in the diagram 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

ElementDefinitionAttribute

Description attribute of the diagram element.

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.

HideInDiagramAttribute

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

ViewInDiagramAttribute

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

Enumerations

SoundChannelMode

An enumeration defining the way a sound is played in a channel.

Default – the channel can play any sounds.

ForbiddenMultiplyBySound – the channel cannot play sounds that are already played.

ForbiddenAnyMultiply – the channel can play only one sound.

SoundGroupPlayMode

Enumeration defining the way of playing of a group of sounds.

Random – select a random sound from the group.

Shuffle – select a random sound from the group without giving repeated values.

Sequentially – sequential selection of sounds from the group.

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 element inspector.

Linear – linear change.

Sin – sinusoidal change.

Cos – cosinusoidal change.

Visual Script Editor

VSEDebug

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

Public properties:

IsBreak – flag indicating that the diagram is stopped

Events:

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

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

Public static methods:

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

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

Call the link tracing in the diagram.

```
void Break()
```

Stop the diagram execution.

`void Next()`

Move to the next diagram element when step-by-step debugging

`void Play()`

Stop the debugging and continue diagram execution in normal mode.

`void Clear()`

Clean the debug data.

Classes

TracertData

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

Properties:

Diagram – diagram identifier.

Link – link identifier.

Data – transmitted data.

VSEditor

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

Classes

InputOutputPointData

A class describing the input and output points of an element

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 elements.

Variables:

Element – reference to an element containing a point.

PointName – the name of a point for connection.

DataType – point data type.

Center – point position.

VSEDiagramWidget

Abstract base class for displaying elements in the diagram editor.

Public properties:

Id – element identifier.

Name – element name.

Comment – element comment.

Description – the description of element type.

WidgetColor – the color of element header.

Instance – the reference to element type instance.

InstanceType – element type.

IsSelected – the flag of element selection in a diagram.

Area – element drawing area.

Protected properties:

isMoved – element dragging flag.

elementStorage – reference to an instance of the data storage class of element.

Protected variables:

mainWidget – main element widget.

headerWidget – element header widget.

descriptionWidget – element type description widget.

bodyWidget – element body widget.

Constructor:

```
VSEDiagramWidget(VSCore.DiagramStorage.ElementsStorage.ElementData
elementStorageData)
```

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

Public methods:

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

Select and deselect the element.

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

Set the new position for the element.

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

Draw an element. 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 element, used if the element 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

WidgetElementContainer

An abstract class that implements the logic of the element 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 element drawing area.

rootContainer – container root widget.

Constructor:

`WidgetElementContainer(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 element widget, contained in the container.

Public virtual properties:

Area – widget drawing area.

Position – widget local position.

WorldPosition – widget world position.

Protected variables:

widgetRect – current element 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 element widget.

BodyWidget

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

HeaderWidget

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

DescriptionWidget

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

VSEElement

A class that implements the logic of drawing a diagram element, except for variables. Inherited from VSEDiagramWidget.

Public properties:

IsMinimized – element minimization flag.

IsHideParameter – flag for hiding a region with element parameters.

IsInversion – element points inversion flag.

IsCanMinimize – flag indicating that the element can be minimized.

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

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

Protected variables:

pointsContainer – container of widgets.

pointsCollapsedContainer – container of widgets in minimized state.

parametersContainer – the root container of element parameters area widgets.

parameterButtonWidget – widget of element parameters button.

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

Public methods:

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

Trace an element.

```
void SetCollapseElement(bool state)
```

Minimize an element.

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

Hide element parameters.

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

Invert the element 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 element parameters values.

Protected virtual methods:

```
void PrepareChildWidget()
```

Prepare element child widgets.

Protected methods:

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

Return data for connection the element 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 element 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 element point widget.

ParametersContainer

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

ParameterValuesContainer

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

ParameterButtonWidget

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

ParameterValueWidget

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

VariableLinkWidget

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

MenuWidget

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

VSEVariable

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

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 element 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.

VSELink

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

Public properties:

Id – link identifier.

SourceElement – a reference to the source element widget.

TargetElement – a reference to the target element 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(DiagramStorage.LinksStorage.LinkData linkStorage, IElement source,
IElement target)
```

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

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 diagram editor.

VSEGroup

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:

```
VSEGroup(DiagramStorage.GroupsStorage.GroupData groupData)
```

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

Public Methods:

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

Draw a group using the mouse cursor lock flag.

```
void AddElement(IBaseWidget widget)
```

Add a widget to a group.

```
bool RemoveElement(IBaseWidget widget)
```

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

VSESelection

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

События:

OnSelectElementChanged – event of changing the currently selected element (selecting a new element)

OnSelectedDiagramChanged – event of changing the currently loaded diagram (loading a new diagram)

OnSelectedDiagramUpdated – event of updating the state of the current diagram.

OnElementSizeChanged – event of changing the size element.

Static variables:

SelectedWidgets – list of currently selected element widgets.

SelectedLinks – list of currently selected links between elements.

UnsavedChanges – counter of unsaved changes in the diagram.

BufferCounter – counter of elements in the buffer.

SceneDiagramController – reference to the instance of the current scene diagram controller.

Public static properties:

CurrentDiagram – the reference to the current diagram storage.

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

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

IsMultipleElements – flag indicating that several elements are selected.

IsMultipleLinks – flag indicating that several links are selected.

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

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

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

IndexOfCurentDiagram – the index of the current open diagram in the list of scene diagrams.

Public static methods:

`void UpdateDiagram()`

Call the update events for the current diagram.

```
void ElementChangedSize(IElement element, float deltaChange)
```

Call the changed size events for the element.

```
void ClearSelection()
```

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

```
void ClearSelectedElements()
```

Clear the list of selected elements.

```
void ClearSelectedLinks()
```

Clear the list of selected links.

```
void ClearSetting()
```

Clear the settings.

```
void ClearAssembly()
```

Clear the assemblies.

```
void ResetUnsavedChanges()
```

Reset unsaved changes counter.

VSETime

Wrapper class for obtaining system time in the diagram 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 diagram editor.

Public static methods:

```
void Start()
```

Initialize the class.

```
void Update()
```

Update the class state.

VSEditorUtils

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

VSEExtension

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

VSEVirtualRect

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.

VSEVirtualPosition

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.

VSEVirualFontSize

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.

Public static methods:

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

Return a set with data about all input points of the element by an instance of the element class. If the element implements ***IInputPointParse*** interface, then a set of names and point types is returned, otherwise a set of names and references to the instances of the FiledInfo class corresponding to the field that describes the input point are returned.

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

The function returns a set with data about all output points of the element by an instance of the element class. If an element implements the ***IOutputPointParse*** interface, then a set of names and class types describing the output point is returned. Otherwise, a set of names and references to the instances of the FiledInfo class corresponding to the field that describes the output point are returned.

```
PropertyInfo GetVariable(Type instanceType)
```

Return an instance of the class that describes the property with the value of the variable by type.

```
List<FieldInfo> GetParameters(Type instanceType)
```

Return a list with instances of classes describing the element parameter fields.

```
void DrawLabelWithAutoTooltip(Rect rect, string label, GUIStyle style)
```

Print text via the ***GUI.Label*** in the style ***style***, with automatic tooltip setting in case the text is outside the borders of the drawing area width. Used to print the names of element points.

```
void DrawLabelWithAutoTooltip(Rect rect, string label, string tooltip, GUIStyle style)
```

Prints text via ***GUI.Label*** in style ***style***, with a tooltip ***tooltip***, and an automatic adding the full text to the tooltip, if the text goes beyond the boundaries of the drawing area width. It is used to print the names of element points that have the Tooltip attribute.

Interfaces

IBaseWidget

Base interface describing the widget of an element in the diagram editor.

Properties:

Id – element identifier from the storage.

Name – element name in the diagram.

Comment – element comment in the diagram.

Description – element class descriptions.

Instance – a reference to the class instance.

InstanceType – element class type.

WidgetColor – element widget header color.

Area – current drawing area of the element widget.

IsSelected – element widget selection flag.

Methods:

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

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

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

Select the element widget.

IElement

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

Properties:

IsMinimized – element minimization state flag.

IsHideParameter – flag indicating the state of element parameters hiding.

IsInversion – element points inversion flag.

IsCanMinimize - flag indicating that element can be minimized.

TracertState – state flag for trace element 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 elements.

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

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

```
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 element. 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 SetCollapseElement(bool state)
```

Setting the element widget minimization state.

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

Set the state of parameters hiding in the element widget.

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

Set the state of points inversion in the element widget.

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

Trace the element in the debug mode.

ILink

An interface describing the link between elements.

Properties:

Id – link identifier from the storage.

SourceElement – the reference to the link source element widget.

TargetElement – the reference to the link target element widget

OutputPoint – the name of output point of source element.

InputPoint – the name of output point of target element.

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 diagram.

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 AddElement(IBaseWidget widget)`

Add a widget to a group.

`bool RemoveElement(IBaseWidget widget)`

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

IParameterContainer

The interface used for the elements 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 element 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 element.

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

VSECustomElementDrawerAttribute

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

Parameters:

ElementType – type of the class of the element whose widget is customized.

Enumerations

PointTypeEnum

Enumeration with a description of element 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.