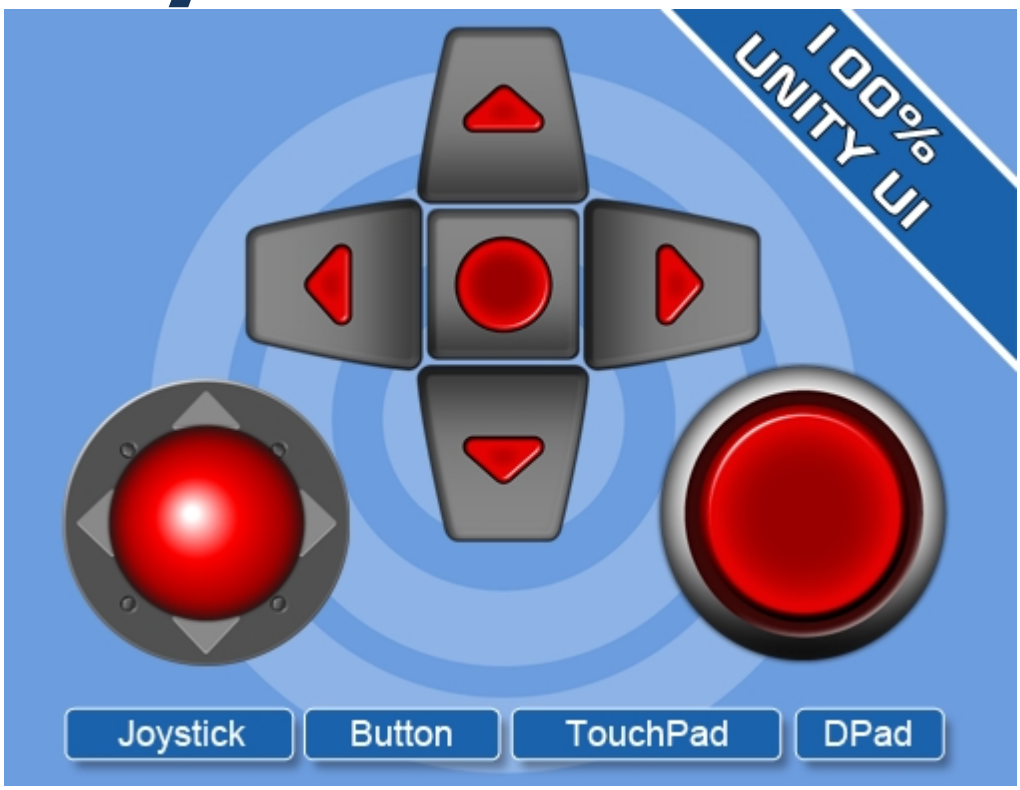
Easy Touch Controls



API Documentation

Table des matières

ETCInput

ETCInput

Description

Interface into the EasyTouch Controls Input system.

Use this class to read/write the axes, buttons, controls.

Axis names are configurable in the inspector and must be unique. The buttons don't have axes, is the name of the button that will be taken into account.

Static functions for controls

SetControlVisible : Set the control identified by ctrlName visible or not.

GetControlVisible : Return true if control the identified by ctrlName is visible.

SetControlActivated : Set the control identified by ctrlName active or not.

GetControlActivated : Return true if control identified by ctrlName is activated.

SetControlSwipeIn : Set the control identified by ctrlName to allow swipe in gesture or not.

GetControlSwipeIn : Return true if the control identified by ctrlName allows swipe in gesture.

SetControlSwipeOut : Set the control identified by ctrlName to allow swipe out gesture or not.

GetControlSwipeOut : Return true if the control identified by ctrlName allows swipe out gesture.

SetDPadAxesCount : Set the number of axes for the DPad identified by ctrlName.

GetDPadAxesCount : Return DPadAxis value for the DPad identified by ctrlName.

Static functions for buttons

GetButtonDown : Returns true during the frame the user pressed down the button identified by buttonName.

GetButton : Returns true while the button identified by buttonName is held down.

GetButtonUp : Returns true the first frame the user releases the button identified by buttonName.

GetButtonValue : Return the value of the button identified by buttonName.

Static functions for axes

GetAxis : Returns the value of the axis identified by axisName.

GetAxisSpeed : Returns the speed value of the axis identified by axisName.

GetAxisDownUp : Returns true during the frame the user move up the axis identified by axisName.

GetAxisDownRight : Returns true during the frame the user move right the axis identified by axisName.

GetAxisDownDown : Returns true during the frame the user move down the axis identified by axisName.

GetAxisDownLeft : Returns true during the frame the user move left the axis identified by axisName.

GetAxisPressedUp : Returns true while the axis identified by axisName is held in up.

GetAxisPressedRight : Returns true while the axis identified by axisName is held in right.

GetAxisPressedDown : Returns true while the axis identified by axisName is held in down.

GetAxisPressedLeft : Returns true while the axis identified by axisName is held in left.

Static functions for axes properties

SetAxisEnabled : Set the axis identified by axisName enabled or not.

GetAxisEnabled : Return true if the axis identified by axisName is enabled.

SetAxisInverted : Set the axis identified by axisName inverted or not.

GetAxisInverted : Return true if the axis identified by axisName is inverted.

SetAxisDeadValue : Set the dead value of the axis identified by axisName.

GetAxisDeadValue : Return the dead value of the axis identified by axisName.

SetAxisSensitivity : Set the sensitivity value of the axis identified by axisName.

GetAxisSensitivity : Return the sensitivity value of the axis identified by axisName.

SetAxisThreshold : Set the threshold value of the axis identified by axisName.

GetAxisThreshold : Return the threshold value of the axis identified by axisName.

SetAxisInertia : Set the inertia enabled or not on the axis identified by axisName.

GetAxisInertia : Return true if the axis identified by axisName has inertia enabled.

SetAxisInertiaSpeed : Set the inertia value of the axis identified by axisName.

GetAxisInertiaSpeed : Return the inertia value of the axis identified by axisName.

SetAxisInertiaThreshold :Set the inertia threshold value of the axis identified by axisName.

GetAxisInertiaThreshold : Return the inertia threshold value of the axis identified by axisName.

SetAxisAutoStabilization: Set the auto-stabilization enabled or not on the axis identified by axisName.

GetAxisAutoStabilization: Return true if the axis identified by axisName has auto-stabilization enabled.

SetAxisAutoStabilizationSpeed : Set the auto-stabilization speed value of the axis identified by axisName.

GetAxisAutoStabilizationSpeed : Return the auto-stabilization speed value of the axis identified by axisName.

SetAxisAutoStabilizationThreshold: Set the auto-stabilization threshold value of the axis identified by axisName.

GetAxisAutoStabilizationThreshold: Return the auto-stabilization threshold value of the axis identified by axisName.

SetAxisClampRotation : Set the clamp rotation enabled or not on the axis identified by axisName.

GetAxisClampRotation : Return true if the axis identified by axisName has clamp rotation enabled.

SetAxisClampRotationValue : Set the min & max clamp rotation value on the axis identified by axisName.

SetAxisClampRotationMinValue : Set the min clamp rotation value on the axis identified by axisName.

GetAxisClampRotationMinValue : Return the min clamp rotation value on the axis identified by axisName.

SetAxisClampRotationMaxValue : Set the max clamp rotation value on the axis identified by axisName.

GetAxisClampRotationMaxValue : Return the max clamp rotation value on the axis identified by axisName.

SetAxisDirecTransform : Set the transform direction action on the axis identified by axisName.

GetAxisDirectTransform : Return the transform direction action on the axis identified by axisName.

SetAxisDirectAction : Set the direction action on the axis identified by axisName.

GetAxisDirectAction : Return the direction action on the axis identified by axisName.

SetAxisAffectedAxis : Set the axis affected by the direct action on the axis identified by axisName.

GetAxisAffectedAxis : Return the axis affected by the direct action on the axis identified by axisName.

Static functions for controls

ETCInput

Description

Interface into the EasyTouch Controls Input system.

Use this class to read/write the axes, buttons, controls.

Axis names are configurable in the inspector and must be unique. The buttons don't have axes, is the name of the button that will be taken into account.

Static functions for controls

SetControlVisible	: Set the control identified by ctrlName visible or not.
GetControlVisible	: Return true if control the identified by ctrlName is visible.
SetControlActivated	: Set the control identified by ctrlName active or not.
GetControlActivated	: Return true if control identified by ctrlName is activated.
SetControlSwipeIn	: Set the control identified by ctrlName to allow swipe in gesture or not.
GetControlSwipeIn	: Return true if the control identified by ctrlName allows swipe in gesture.
SetControlSwipeOut	: Set the control identified by ctrlName to allow swipe out gesture or not.
GetControlSwipeOut	: Return true if the control identified by ctrlName allows swipe out gesture.
SetDPadAxesCount	: Set the number of axes for the DPad identified by ctrlName.
GetDPadAxesCount	: Return DPadAxis value for the DPad identified by ctrlName.

SetControlVisible

ETCInput.SetControlVisible

public static void **SetControlVisible**(string: ctrlName, bool value)

Description

Set the control identified by ctrlName visible or not, relative to value parameter.

```
using UnityEngine;
using System.Collections;

public class ExampleClass : MonoBehaviour {
```

```
    void Start() {
        ETCInput.SetControlVisible("MyJoystick",false);
    }
}
```

GetControlVisible

ETCInput.GetControlVisible

public static void **GetControlVisible**(string: ctrlName)

Description

Return true if control identified by ctrlName is visible.

```
using UnityEngine;
using System.Collections;

public class ExampleClass : MonoBehaviour {

    void Update() {
        if (ETCInput.GetControlVisible("MyJoystick")) {
            Debug.Log( "Mys joystick is visible");
        }
    }
}
```

SetControlActivated

ETCInput.SetControlActivated

public static void **SetControlActivated**(string: ctrlName, bool value)

Description

Set the control identified by ctrlName active or not, relative to value parameter.

```
using UnityEngine;
using System.Collections;

public class ExampleClass : MonoBehaviour {

    void Start() {
        ETCInput.SetControlActivated("MyJoystick",true);
    }
}
```

GetControlActivated

ETCInput.GetControlActivated

public static void **GetControlActivated**(string: ctrlName)

Description

Return true if control identified by ctrlName is activated.

```
using UnityEngine;
using System.Collections;

public class ExampleClass : MonoBehaviour {

    void Update() {
        if (ETCInput.GetControlActivated("MyJoystick")) {
            Debug.Log( "My joystick is activated");
        }
    }
}
```

SetControlSwipeIn

ETCInput.SetControlSwipeIn

public static void **SetControlSwipeIn**(string: ctrlName, bool value)

Description

Set the control identified by ctrlName to allow swipein gesture or not, relative to value parameter.

Swipe in allows you to use the control even if the start touch occurs outside the control and slid over him.This behavior is available only for the Touchpad & buttons.

```
using UnityEngine;
using System.Collections;

public class ExampleClass : MonoBehaviour {

    void Start() {
        ETCInput.SetControlSwipeIn("MyButton",true);
    }
}
```

GetControlSwipeIn

ETCInput.GetControlSwipeIn

public static void **GetControlSwipeIn**(string: ctrlName)

Description

Return true if the control identified by ctrlName allows swipein behavior.

Swipe in allows you to use the control even if the start touch occurs outside the control and slid over him.This behavior is available only for the Touchpad & buttons.

```
using UnityEngine;
using System.Collections;

public class ExampleClass : MonoBehaviour {

    void Update() {
        if (ETCInput.GetControlSwipeIn("MyButton")) {
            Debug.Log( "My Button has swipeIn enabled");
        }
    }
}
```

SetControlSwipeOut

ETCInput.SetControlSwipeOut

public static void **SetControlSwipeOut**(string: ctrlName, bool value)

Description

Set the control identified by ctrlName to allow swipeout gesture or not relative to value parameter.

Swipe Out allows you to use the control even if the current touch position isn't over him but the touch start occurs over him.This behavior is available only for the Touchpad & buttons.

```
using UnityEngine;
using System.Collections;

public class ExampleClass : MonoBehaviour {

    void Start() {
        ETCInput.SetControlSwipeOut("MyButton",true);
    }
}
```

GetControlSwipeOut

ETCInput.GetControlSwipeOut

public static void **GetControlSwipeOut**(string: ctrlName)

Description
Return true if the control identified by ctrlName allows swipeout behavior.

Swipe in allows you to use the control even if the start touch occurs outside the control and slid over him.This behavior is available only for the Touchpad & buttons.

```
using UnityEngine;
using System.Collections;

public class ExampleClass : MonoBehaviour {

    void Update() {
        if (ETCInput.GetControlSwipeOut("MyButton")) {
            Debug.Log( "My Button has swipeOut enabled");
        }
    }
}
```

SetDPadAxesCount

ETCInput.SetDPadAxesCount

public static void **SetDPadAxesCount**(string: ctrlName, ETCBase.DPadAxis value)

Description

Set the number of axes for the DPad identified by ctrlName relative to value parameter.

The DPad can be used with 2 or 4 axes:

- 2 axes : Up / Down & Left / Right
- 4 axes : Up / Down & Left / Right & Up Left /Right & Down Left/Right

```
using UnityEngine;
using System.Collections;

public class ExampleClass : MonoBehaviour {

    void Start() {
        ETCInput.SetDPadAxesCount("MyDPad",ETCBase.DPadAxis.Two_Axis);
    }
}
```

GetDPadAxesCount

ETCInput.GetDPadAxesCount

public static ETCBase.DPadAxis **GetDPadAxesCount**(string: ctrlName)

Description

Return DPadAxis value for the DPad identified by ctrlName.

The DPad can be used with 2 or 4 axes:

- 2 axes : Up / Down & Left / Right
- 4 axes : Up / Down & Left / Right & Up Left /Right & Down Left/Right

```
using UnityEngine;
using System.Collections;

public class ExampleClass : MonoBehaviour {

    void Update() {
        if (ETCInput.GetDPadAxesCounr("MyDPad") == ETCBase.DPadAxis.Two_Axis) {
            Debug.Log( "My DPad has 2 axes");
        }
    }
}
```

Static functions for buttons

ETCInput

Description

Interface into the EasyTouch Controls Input system.

Use this class to read/write the axes, buttons, controls.

Axis names are configurable in the inspector and must be unique. The buttons don't have axes, is the name of the button that will be taken into account.

Static functions for buttons

GetButtonDown : Returns true during the frame the user pressed down the button identified by buttonName.

GetButton : Returns true while the button identified by buttonName is held down.

GetButtonUp : Returns true the first frame the user releases the button identified by buttonName.

GetButtonValue : Return the value of the button identified by buttonName.

GetButtonDown

ETCInput.GetButtonDown

public static bool **GetButtonDown**(string: buttonName)

Description

Returns true during the frame the user pressed down the virtual button identified by buttonName.

```
using UnityEngine;
using System.Collections;

public class ExampleClass : MonoBehaviour {
    public GameObject projectile;

    void Update() {
        if (ETCInput.GetButtonDown("Fire1") ) {
            Instantiate(projectile, transform.position, transform.rotation) as GameObject;
        }
    }
}
```

GetButton

ETCInput.GetButton

public static bool **GetButton**(string: buttonName)

Description

Returns true while the button identified by buttonName is held down.

```
using UnityEngine;
using System.Collections;

public class ExampleClass : MonoBehaviour {

    public GameObject projectile;
    public float fireRate = 0.5F;
    private float nextFire = 0.0F;

    void Update() {
        if (ETCInput.GetButton("Fire1") && Time.time > nextFire) {
            nextFire = Time.time + fireRate;
            Instantiate(projectile, transform.position, transform.rotation) as GameObject;
        }
    }
}
```

GetButtonUp

ETCInput.GetButtonUp

public static bool **GetButtonUp**(string: buttonName)

Description

Returns true the first frame the user releases the virtual button identified by buttonName

```
using UnityEngine;
using System.Collections;

public class ExampleClass : MonoBehaviour {

    public GameObject projectile;
    public float fireRate = 0.5F;
    private float nextFire = 0.0F;

    void Update() {
        if (ETCInput.GetButtonUp("Fire1") && Time.time > nextFire) {
            nextFire = Time.time + fireRate;
            Instantiate(projectile, transform.position, transform.rotation) as GameObject;
        }
    }
}
```

GetButtonValue

ETCInput.GetButtonValue

public static float **GetButtonUp**(string: buttonName)

Description

Return the value of the button identified by buttonName.

Uses this function when your button has over time enabled, to know his value.

```
using UnityEngine;
using System.Collections;

public class ExampleClass : MonoBehaviour {

    public GameObject projectile;
    public float fireRate = 0.5F;

    void Update() {
        if (ETCInput.GetButtonValue("Fire1")  > 0.5f) {
            Instantiate(projectile, transform.position, transform.rotation) as GameObject;
        }
    }
}
```

Static functions for axes

ETCInput

Description
Interface into the EasyTouch Controls Input system.

Use this class to read/write the axes, buttons, controls.

Axis names are configurable in the inspector and must be unique. The buttons don't have axes, is the name of the button that will be taken into account.

Static functions for axes

GetAxis	: Returns the value of the axis identified by axisName.
GetAxisSpeed	: Returns the speed value of the axis identified by axisName.
GetAxisDownUp	: Returns true during the frame the user move up the axis identified by axisName.
GetAxisDownRight	: Returns true during the frame the user move right the axis identified by axisName.
GetAxisDownDown	: Returns true during the frame the user move down the axis identified by axisName.
GetAxisDownLeft	: Returns true during the frame the user move left the axis identified by axisName.
GetAxisPressedUp	: Returns true while the axis identified by axisName is held in up.
GetAxisPressedRight	: Returns true while the axis identified by axisName is held in right.
GetAxisPressedDown	: Returns true while the axis identified by axisName is held in down.
GetAxisPressedLeft	: Returns true while the axis identified by axisName is held in left.

GetAxis

ETCInput.GetAxis

public static float **GetAxis**(string: axisName)

Description
Returns the value of the virtual axis identified by axisName.

The value will be in the range -1...1 for Joystick, DPad. If the axis belongs to a TouchPad the range is not -1...1, because this function return the delta position.

```
using UnityEngine;
```

```
using System.Collections;

public class ExampleClass : MonoBehaviour {

    public float speed = 10.0F;
    public float rotationSpeed = 100.0F;

    void Update() {
        float translation = ETCInput.GetAxis("Vertical") * speed;
        float rotation = ETCInput.GetAxis("Horizontal") * rotationSpeed;
        translation *= Time.deltaTime;
        rotation *= Time.deltaTime;
        transform.Translate(0, 0, translation);
        transform.Rotate(0, rotation, 0);
    }

}
```

GetAxisSpeed

ETCInput.GetAxisSpeed

public static float **GetAxisSpeed**(string: axisName)

Description

Returns the speed value of the virtual axis identified by axisName.

The value will be the equal to =Axis Value * Axis Sensitivity * Time.DeltaTime.

Axis Sensitivity = Axis speed for Joystick and DPad on inspector.

```
using UnityEngine;
using System.Collections;

public class ExampleClass : MonoBehaviour {

    void Update() {
        float translation = ETCInput.GetAxisSpeed("Vertical") ;
        float rotation = ETCInput.GetAxisSpeed("Horizontal") ;
        transform.Translate(0, 0, translation);
        transform.Rotate(0, rotation, 0);
    }

}
```

GetAxisDownUp

ETCInput.GetAxisDownUp

public static bool **GetAxisDownUp**(string: buttonName)

Description

Returns true during the frame the user move up the axis identified by axisName.

For Joystick the value of its axis must be> axis threshold (On / Off threshold in the inspector)

```
using UnityEngine;
using System.Collections;

public class ExampleClass : MonoBehaviour {

    void Update() {
        if (ETCInput.GetAxisDownUp("Vertical") ) {
            Jump();
        }
    }
}
```

GetAxisDownRight

ETCInput.GetAxisDownRight

public static bool **GetAxisDownRight**(string: buttonName)

Description

Returns true during the frame the user move right the axis identified by axisName.

For Joystick the value of its axis must be greater than axis threshold (On / Off threshold in the inspector)

```
using UnityEngine;
using System.Collections;

public class ExampleClass : MonoBehaviour {

    void Update() {
        if (ETCInput.GetAxisDownRight("Horizontal") ) {
            MoveOneStepRight();
        }
    }
}
```

GetAxisDownDown

ETCInput.GetAxisDownDown

public static bool **GetAxisDownDown**(string: buttonName)

Description

Returns true during the frame the user move down the axis identified by axisName.

For Joystick the value of its axis must be> axis threshold (On / Off threshold in the inspector)

```
using UnityEngine;
using System.Collections;

public class ExampleClass : MonoBehaviour {

        void Update() {
        if (ETCInput.GetAxisDownDown("Vertical") ) {
            Crush();
        }
    }
}
```

GetAxisDownLeft

ETCInput.GetAxisDownLeft

public static bool **GetAxisDownLeft**(string: buttonName)

Description

Returns true during the frame the user move left the axis identified by axisName.

For Joystick the value of its axis must be greater than axis threshold (On / Off threshold in the inspector)

```
using UnityEngine;
using System.Collections;

public class ExampleClass : MonoBehaviour {

    void Update() {
        if (ETCInput.GetAxisDownLeft("Horizontal") ) {
            MoveOneStepLeft();
        }
    }
}
```

GetAxisPressedUp

ETCInput.GetAxisPressedUp

public static bool **GetAxisPressedUp**(string: buttonName)

Description

Returns true while the axis identified by axisName is held in up.

For Joystick the value of its axis must be> axis threshold one time (On / Off threshold in the inspector)

```
using UnityEngine;
using System.Collections;

public class ExampleClass : MonoBehaviour {

    void Update() {
        if (ETCInput.GetAxisPressedUp("Vertical") ) {
            transform.Translate(Vector3.forward * Time.deltaTime);
        }
    }
}
```

GetAxisPressedRight

ETCInput.GetAxisPressedRight

public static bool **GetAxisPressedRight**(string: buttonName)

Description

Returns true while the axis identified by axisName is held in right.

For Joystick the value of its axis must be> axis threshold one time (On / Off threshold in the inspector)

```
using UnityEngine;
using System.Collections;

public class ExampleClass : MonoBehaviour {

    void Update() {
        if (ETCInput.GetAxisPressedUp("Horizontal") ) {
            transform.Translate(Vector3.right * Time.deltaTime);
        }
    }
}
```

GetAxisPressedDown

ETCInput.GetAxisPressedDown

public static bool **GetAxisPressedDown**(string: buttonName)

Description

Returns true while the axis identified by axisName is held in down.

For Joystick the value of its axis must be> axis threshold one time (On / Off threshold in the inspector)

```
using UnityEngine;
using System.Collections;

public class ExampleClass : MonoBehaviour {

    void Update() {
        if (ETCInput.GetAxisPressedDown("Vertical") ) {
            transform.Translate(Vector3.down * Time.deltaTime);
        }
    }
}
```

GetAxisPressedLeft

ETCInput.GetAxisPressedLeft

public static bool **GetAxisPressedLeft**(string: buttonName)

Description

Returns true while the axis identified by axisName is held in left.

For Joystick the value of its axis must be> axis threshold one time (On / Off threshold in the inspector)

```
using UnityEngine;
using System.Collections;

public class ExampleClass : MonoBehaviour {

    void Update() {
        if (ETCInput.GetAxisPressedUp("Horizontal") ) {
            transform.Translate(Vector3.left * Time.deltaTime);
        }
    }
}
```

Static functions for axes properties

ETCInput

Description

Interface into the EasyTouch Controls Input system.

Use this class to read/write the axes, buttons, controls.

Axis names are configurable in the inspector and must be unique. The buttons don't have axes, is the name of the button that will be taken into account.

Static functions for axes properties

SetAxisEnabled : Set the axis identified by axisName enabled or not.

GetAxisEnabled : Return true if the axis identified by axisName is enabled.

SetAxisInverted : Set the axis identified by axisName inverted or not.

GetAxisInverted : Return true if the axis identified by axisName is inverted.

SetAxisDeadValue : Set the dead value of the axis identified by axisName.

GetAxisDeadValue : Return the dead value of the axis identified by axisName.

SetAxisSensitivity : Set the sensitivity value of the axis identified by axisName.

GetAxisSensitivity : Return the sensitivity value of the axis identified by axisName.

SetAxisThreshold : Set the threshold value of the axis identified by axisName.

GetAxisThreshold : Return the threshold value of the axis identified by axisName.

SetAxisInertia : Set the inertia enabled or not on the axis identified by axisName.

GetAxisInertia : Return true if the axis identified by axisName has inertia enabled.

SetAxisInertiaSpeed : Set the inertia value of the axis identified by axisName.

GetAxisInertiaSpeed : Return the inertia value of the axis identified by axisName.

SetAxisInertiaThreshold :Set the inertia threshold value of the axis identified by axisName.

GetAxisInertiaThreshold : Return the inertia threshold value of the axis identified by axisName.

SetAxisAutoStabilization: Set the auto-stabilization enabled or not on the axis identified by axisName.

GetAxisAutoStabilization: Return true if the axis identified by axisName has auto-stabilization enabled.

SetAxisAutoStabilizationSpeed : Set the auto-stabilization speed value of the axis identified by axisName.

GetAxisAutoStabilizationSpeed : Return the auto-stabilization speed value of the axis identified by

axisName.

SetAxisAutoStabilizationThreshold: Set the auto-stabilization threshold value of the axis identified by axisName.

GetAxisAutoStabilizationThreshold: Return the auto-stabilization threshold value of the axis identified by axisName.

SetAxisClampRotation : Set the clamp rotation enabled or not on the axis identified by axisName.

GetAxisClampRotation : Return true if the axis identified by axisName has clamp rotation enabled.

SetAxisClampRotationValue : Set the min & max clamp rotation value on the axis identified by axisName.

SetAxisClampRotationMinValue : Set the min clamp rotation value on the axis identified by axisName.

GetAxisClampRotationMinValue : Return the min clamp rotation value on the axis identified by axisName.

SetAxisClampRotationMaxValue : Set the max clamp rotation value on the axis identified by axisName.

GetAxisClampRotationMaxValue : Return the max clamp rotation value on the axis identified by axisName.

SetAxisDirecTransform : Set the transform direction action on the axis identified by axisName.

GetAxisDirectTransform : Return the transform direction action on the axis identified by axisName.

SetAxisDirectAction : Set the direction action on the axis identified by axisName.

GetAxisDirectAction : Return the direction action on the axis identified by axisName.

SetAxisAffectedAxis : Set the axis affected by the direct action on the axis identified by axisName.

GetAxisAffectedAxis : Return the axis affected by the direct action on the axis identified by axisName.

SetAxisEnabled

ETCInput.SetAxisEnabled

public static void **SetAxisEnabled**(string: axisName, bool value)

Description

Set the axis identified by axisName enabled or not, relative to value parameter.

```
using UnityEngine;
using System.Collections;

public class ExampleClass : MonoBehaviour {

    void Start() {
        ETCInput.SetAxisEnabled("Vertical",true);
    }
}
```

GetAxisEnabled

ETCInput.GetAxisEnabled

public static bool **GetAxisEnabled**(string: axisName)

Description

Return true if the axis identified by axisName is enabled.

```
using UnityEngine;
using System.Collections;

public class ExampleClass : MonoBehaviour {

    void Update() {
        if (ETCInput.GetAxisEnabled("Horizontal")) {
            Debug.Log( "Axis Horizontal is enabled");
        }
    }
}
```

SetAxisInverted

ETCInput.SetAxisInverted

public static void **SetAxisInverted**(string: axisName, bool value)

Description

Set the axis identified by axisName inverted or not.

```
using UnityEngine;
using System.Collections;

public class ExampleClass : MonoBehaviour {

    void Start() {
        ETCInput.SetAxisInverted("Vertical",true);
    }
}
```

GetAxisInverted

ETCInput.GetAxisInverted

public static bool **GetAxisInverted**(string: axisName)

Description

Return true if the axis identified by axisName is inverted.

```
using UnityEngine;
using System.Collections;

public class ExampleClass : MonoBehaviour {

    void Update() {
        if (ETCInput.GetAxisInverted("Horizontal")) {
            Debug.Log( "Axis Horizontal is inverted");
        }
    }
}
```

SetAxisDeadValue

ETCInput.SetAxisDeadValue

public static void **SetAxisDeadValue**(string: axisName, float value)

Description

Set the dead value of the axis identified by axisName.

This value corresponds to a dead zone in relative value (0..1), where the axis will not be considered in motion (only for joystick)

```
using UnityEngine;
using System.Collections;

public class ExampleClass : MonoBehaviour {

    void Start() {
        ETCInput.SetAxisDeadValue("Vertical",0.2f);
    }
}
```

GetAxisDeadValue

ETCInput.GetAxisDeadValue

public static float **GetAxisDeadValue**(string: axisName)

Description

Return the dead value of the axis identified by axisName.

This value corresponds to a dead zone in relative value (0..1), where the axis will not be considered in motion (only for joystick)

```
using UnityEngine;
using System.Collections;

public class ExampleClass : MonoBehaviour {

    void Update() {
        float deadV = ETCInput.GetAxisDeadValue("Horizontal");
         Debug.Log( deadV);
    }
}
```

SetAxisSensitivity

ETCInput.SetAxisSensitivity

public static void **SetAxisSensitivity**(string: axisName, float value)

Description

Set the sensitivity value of the axis identified by axisName.

This value is used by direct mode to operate the operation, and in the calculation of value returned by ETCInput.GetAxisSpeed.

Axis Sensitivity = Axis speed for Joystick and DPad on inspector.

```
using UnityEngine;
using System.Collections;

public class ExampleClass : MonoBehaviour {

    void Start() {
        ETCInput.SetAxisSensitivity("Vertical",50f);
    }
}
```

GetAxisSensitivity

ETCInput.GetAxisSensitivity

public static float **GetAxisSensitivity**(string: axisName)

Description

Return the sensitivity value of the axis identified by axisName.

This value is used by direct mode to operate the operation, and in the calculation of value returned by ETCInput.GetAxisSpeed.

Axis Sensitivity = Axis speed for Joystick and DPad on inspector.

```
using UnityEngine;
using System.Collections;

public class ExampleClass : MonoBehaviour {

    void Update() {
        float sensitivity = ETCInput.GetAxisSensitivity("Horizontal");
         Debug.Log( sensitivity );
    }
}
```

SetAxisThreshold

ETCInput.SetAxisThreshold

public static void **SetAxisThreshold**(string: axisName, float value)

Description

Set the threshold value of the axis identified by axisName.

This value is used to determine the threshold when the axis will be considered down for the first time.

```
using UnityEngine;
using System.Collections;

public class ExampleClass : MonoBehaviour {

    void Start() {
        ETCInput.SetAxisThreshold("Vertical",0.2f);
    }
}
```

GetAxisThreshold

ETCInput.GetAxisThreshold

public static float **GetAxisThreshold**(string: axisName)

Description

Return the threshold value of the axis identified by axisName.

This value is used to determine the threshold when the axis will be considered down for the first time.

```
using UnityEngine;
using System.Collections;

public class ExampleClass : MonoBehaviour {

    void Update() {
        float threshold = ETCInput.GetAxisThreshold("Horizontal");
         Debug.Log( threshold );
    }
}
```

SetAxisInertia

ETCInput.SetAxisInertia

public static void **SetAxisInertia**(string: axisName, bool value)

Description

Set the inertia enabled or not on the axis identified by axisName.

```
using UnityEngine;
using System.Collections;

public class ExampleClass : MonoBehaviour {

    void Start() {
        ETCInput.SetAxisInertia("Vertical",true);
    }
}
```

GetAxisInertia

ETCInput.GetAxisInertia

public static bool **GetAxisInertia**(string: axisName)

Description

Return true if the axis identified by axisName has inertia enabled.

This value is used to determine the threshold when the axis will be considered down for the first time.

```
using UnityEngine;
using System.Collections;

public class ExampleClass : MonoBehaviour {

    void Update() {
        if ( ETCInput.GetAxisInertia("Horizontal")){
            Debug.Log( "Inertia is enabled" );
        }
    }
}
```

SetAxisInertiaSpeed

ETCInput.SetAxisInertiaSpeed

public static void **SetAxisInertiaSpeed**(string: axisName, float value)

Description

Set the inertia value of the axis identified by axisName.

The higher the value is, the greater the inertia effect is important.

```
using UnityEngine;
using System.Collections;

public class ExampleClass : MonoBehaviour {

    void Start() {
        ETCInput.SetAxisInertiaSpeed("Vertical",200);
    }
}
```

GetAxisInertiaSpeed

ETCInput.GetAxisInertiaSpeed

public static float **GetAxisInertiaSpeed**(string: axisName)

Description

Return the inertia value of the axis identified by axisName.

The higher the value is, the greater the inertia effect is important.

```
using UnityEngine;
using System.Collections;

public class ExampleClass : MonoBehaviour {

    void Update() {
        float inertiaSpeed =  ETCInput.GetAxisInertiaSpeed("Horizontal");
        Debug.Log( inertiaSpeed );

    }
}
```

SetAxisInertiaThreshold

ETCInput.SetAxisInertiaThreshold

public static void **SetAxisInertiaThresold**(string: axisName, float value)

Description

Set the inertia threshold value of the axis identified by axisName.

This value allows to determine the threshold below which the axis will be reset to the 0 position when inertia is enabled.

```
using UnityEngine;
using System.Collections;

public class ExampleClass : MonoBehaviour {

    void Start() {
        ETCInput.SetAxisInertiaThreshold("Vertical",0.1f);
    }
}
```

GetAxisInertiaThreshold

ETCInput.GetAxisInertiaThreshold

public static float **GetAxisInertiaThreshold**(string: axisName)

Description

Return the inertia threshold value of the axis identified by axisName.

This value allows to determine the threshold below which the axis will be reset to the 0 position when inertia is enabled.

```
using UnityEngine;
using System.Collections;

public class ExampleClass : MonoBehaviour {

    void Update() {
        float inertiaThreshold =  ETCInput.GetAxisInertiaThreshold("Horizontal");
        Debug.Log( inertiaThreshold );

    }
}
```

SetAxisAutoStabilization

ETCInput.SetAxisAutoStabilization

public static void **SetAxisAutoStabilization**(string: axisName, bool value)

Description

Set the auto-stabilization enabled or not on the axis identified by axisName.

Self-stabilization is take into account only in the direct mode with an action on Local Rotation.It allows to return the object to its original rotation.

```
using UnityEngine;
using System.Collections;

public class ExampleClass : MonoBehaviour {

    void Start() {
        ETCInput.SetAxisAutoStabilization("Vertical",true);
    }
}
```

GetAxisAutoStabilization

ETCInput.GetAxisAutoStabilization

public static bool **GetAxisAutoStabilization**(string: axisName)

Description

Return true if the axis identified by axisName has auto-stabilization enabled.

Self-stabilization is take into account only in the direct mode with an action on Local Rotation.It allows to return the object to its original rotation.

```
using UnityEngine;
using System.Collections;

public class ExampleClass : MonoBehaviour {

    void Update() {
        if ( ETCInput.GetAxisAutoStabilization("Horizontal")){
            Debug.Log("AutoStabilization is enabled );
        }
    }
}
```

SetAxisAutoStabilizationSpeed

ETCInput.SetAxisAutoStabilizationSpeed

public static void **SetAxisAutoStabilization**(string: axisName, float value)

Description

Set the auto-stabilization speed value of the axis identified by axisName.

Self-stabilization is take into account only in the direct mode with an action on Local Rotation.It allows to return the object to its original rotation.

```
using UnityEngine;
using System.Collections;

public class ExampleClass : MonoBehaviour {

    void Start() {
        ETCInput.SetAxisAutoStabilizationSpeed("Vertical",200);
    }
}
```

GetAxisAutoStabilizationSpeed

ETCInput.GetAxisAutoStabilizationSpeed

public static float **GetAxisAutoStabilizationSpeed**(string: axisName)

Description

Return the auto-stabilization speed value of the axis identified by axisName.

Self-stabilization is take into account only in the direct mode with an action on Local Rotation.It allows to return the object to its original rotation.

```
using UnityEngine;
using System.Collections;

public class ExampleClass : MonoBehaviour {

    void Update() {
        float speed = ETCInput.GetAxisAutoStabilizationSpeed("Horizontal");
        Debug.Log(Speed );
    }
}
```

SetAxisAutoStabilizationThreshold

ETCInput.SetAxisAutoStabilizationThresold

public static void **SetAxisAutoStabilizationThreshold**(string: axisName, float value)

Description

Set the auto-stabilization threshold value of the axis identified by axisName.

This value allows to determine the threshold below which the axis will be reset to the 0 position when auto-stabilization is enabled.
Self-stabilization is take into account only in the direct mode with an action on Local Rotation.It allows to return the object to its original rotation.

```
using UnityEngine;
using System.Collections;

public class ExampleClass : MonoBehaviour {

    void Start() {
        ETCInput.SetAxisAutoStabilizationThreshold("Vertical",0.1f);
    }
}
```

GetAxisAutoStabilizationThreshold

ETCInput.GetAxisAutoStabilizationThreshold

public static float **GetAxisAutoStabilizationThreshold**(string: axisName)

Description

Return the auto-stabilization threshold value of the axis identified by axisName..

Self-stabilization is take into account only in the direct mode with an action on Local Rotation.It allows to return the object to its original rotation.

```
using UnityEngine;
using System.Collections;

public class ExampleClass : MonoBehaviour {

    void Update() {
        float threshold = ETCInput.GetAxisAutoStabilizationThreshold("Horizontal");
        Debug.Log(Speed );
    }
}
```

SetAxisClampRotation

ETCInput.SetAxisClampRotation

public static void **SetAxisClampRotation**(string: axisName, bool value)

Description

Set the clamp rotation enabled or not on the axis identified by axisName.

ClampRotation is take into account only in the direct mode with an action on Local Rotation.

```
using UnityEngine;
using System.Collections;

public class ExampleClass : MonoBehaviour {

    void Start() {
        ETCInput.SetAxisClampRotation("Vertical",true);
    }
}
```

GetAxisClampRotation

ETCInput.GetAxisClampRotation

public static bool **GetAxisClampRotation**(string: axisName)

Description

Return true if the axis identified by axisName has clamp rotation enabled.

ClampRotation is take into account only in the direct mode with an action on Local Rotation.

```
using UnityEngine;
using System.Collections;

public class ExampleClass : MonoBehaviour {

    void Update() {
        if ( ETCInput.GetAxisClampRotation("Horizontal")){
            Debug.Log("ClampRotation is enabled );
        }
    }
}
```

SetAxisClampRotationValue

ETCInput.SetAxisClampRotationValue

public static void **SetAxisClampRotationValue**(string: axisName, float min , float max)

Description

Set the min & max clamp rotation value on the axis identified by axisName.

Horizontal Axis
min = limit angle to the left
max = limit angle to the right

VertivalAxis
min = limit angle up
max = limit angle down

```
using UnityEngine;
using System.Collections;

public class ExampleClass : MonoBehaviour {

    void Start() {
        ETCInput.SetAxisClampRotationValue("Vertical",60,15);
    }
}
```

SetAxisClampRotationMinValue

ETCInput.SetAxisClampRotationMinValue

public static void **SetAxisClampRotationMinValue**(string: axisName, float value)

Description

Set the min rotation value on the axis identified by axisName.

Horizontal Axis
min = limit angle to the left

VertivalAxis
min = limit angle up

```
using UnityEngine;
using System.Collections;

public class ExampleClass : MonoBehaviour {

    void Start() {
        ETCInput.SetAxisClampRotationMinValue("Vertical",60);
    }
}
```

GetAxisClampRotationMinValue

ETCInput.GetAxisClampRotationMinValue

public static float **GetAxisClampRotationMinValue**(string: axisName)

Description

Return the min clamp rotation value on the axis identified by axisName.

Horizontal Axis
min = limit angle to the left

VertivalAxis
min = limit angle up

```
using UnityEngine;
using System.Collections;

public class ExampleClass : MonoBehaviour {

    void Update() {
        float min = ETCInput.GetAxisClampRotationMinValue("Horizontal");
        Debug.Log(min );
    }
}
```

SetAxisClampRotationMaxValue

ETCInput.SetAxisClampRotationMaxValue

public static void **SetAxisClampRotationMaxValue**(string: axisName, float value)

Description

Set the max rotation value on the axis identified by axisName.

Horizontal Axis
max = limit angle to the right

VertivalAxis
max = limit angle down

```
using UnityEngine;
using System.Collections;

public class ExampleClass : MonoBehaviour {

    void Start() {
        ETCInput.SetAxisClampRotationMaxValue("Vertical",60);
    }
}
```

GetAxisClampRotationMaxValue

ETCInput.GetAxisClampRotationMaxValue

public static float **GetAxisClampRotationMaxValue**(string: axisName)

Description

Return the max clamp rotation value on the axis identified by axisName.

Horizontal Axis
max = limit angle to the right

VertivalAxis
max = limit angle down

```
using UnityEngine;
using System.Collections;

public class ExampleClass : MonoBehaviour {

    void Update() {
        float max = ETCInput.GetAxisClampRotationMaxValue("Horizontal");
        Debug.Log(max );
    }
}
```

SetAxisDirecTransform

ETCInput.SetAxisDirectTransform

public static void **SetAxisDirectTransform**(string: axisName, Transform value)

Description

Set the transform direction action on the axis identified by axisName.

```
using UnityEngine;
using System.Collections;

public class ExampleClass : MonoBehaviour {

    public GameObject object2Move;

    void Start() {
        ETCInput.SetAxisDirectTransform("Vertical",object2Move.transform);
    }
}
```

GetAxisDirectTransform

ETCInput.GetAxisDirectTransform

public static Transform **GetAxisDirectTransform**(string: axisName)

Description

Return the transform direction action on the axis identified by axisName.

```
using UnityEngine;
using System.Collections;

public class ExampleClass : MonoBehaviour {

    void Update() {
        Transform obj2move = ETCInput.GetAxisDirectTransform("Horizontal");
        Debug.Log( obj2move.name );
    }
}
```

SetAxisDirectAction

ETCInput.SetAxisDirectAction

public static void **SetAxisDirectAction**(string: axisName, ETCAxis.DirectAction value)

Description

Set the direction action on the axis identified by axisName.

```
using UnityEngine;
using System.Collections;

public class ExampleClass : MonoBehaviour {

    void Start() {
        ETCInput.SetAxisDirectAction("Vertical",ETCAxis.DirectAction.RelativeForce);
    }
}
```

GetAxisDirectAction

ETCInput.GetAxisDirectAction

public static ETCAxis.DirectAction **GetAxisDirectAction**(string: axisName)

Description

Return the direction action on the axis identified by axisName.

```
using UnityEngine;
using System.Collections;

public class ExampleClass : MonoBehaviour {

    void Update() {
        if (ETCInput.GetAxisDirectAction("Horizontal") == ETCAxis.DirectAction.Rotate){
            Debug.Log( "Rotation" );
        }
    }
}
```

SetAxisAffectedAxis

ETCInput.SetAxisAffectedAxis

public static void **SetAxisAffectedAxis**(string: axisName, ETCAxis.AxisInfluenced value)

Description

Set the axis affected by the direct action on the axis identified by axisName.

```
using UnityEngine;
using System.Collections;

public class ExampleClass : MonoBehaviour {

    void Start() {
        ETCInput.SetAxisAffectedAxis("Vertical",ETCAxis.AxisInfluenced.X);
    }
}
```

GetAxisAffectedAxis

ETCInput.GetAxisAffectedAxis

public static ETCAxis.AxisInfluenced **GetAxisAffectedAxis**(string: axisName)

Description

Return the axis affected by the direct action on the axis identified by axisName.

```
using UnityEngine;
using System.Collections;

public class ExampleClass : MonoBehaviour {

    void Update() {
        if (ETCInput.GetAxisAffectedAxis("Horizontal") == ETCAxis.AxisInfluenced.X){
            Debug.Log( 'Direction action on X axis" );
        }
    }
}
```

Enumerations

DirectAction

ETCAxis.DirectAction

Description
Identified the type of action that will be applied to the object that is in direct mode

Variables

Rotate

RotateLocal

Translate

TranslateLocal

Scale

Force

RelativeForce

Torque

RelativeTorque

DPadAxis

ETCBase.DPadAxis

Description
Identified the axes count for a DPad

Variables

Two_Axis

Four_Axis

AxisInfluenced

ETCAxis.AxisInfluenced

Description
Identified the axis of action that will be applied to the object that is in direct mode.

Variables

X

Y

Z