Adaptive Hands

1.2.2

Generated by Doxygen 1.9.5

Chapter 1

Namespace Index

1.1 Package List

Here are the packages with brief descriptions (if available):

Chapter 2

Hierarchical Index

2.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

Chapter 3

Class Index

3.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 4

Namespace Documentation

4.1 AdaptiveHands Namespace Reference

Classes

- class FingerBoneTransforms

  A class that holds Transform references to all possible bones in a finger.
- class HandBoneTransforms

  A class that holds Transform references to all possible bones in a hand.
- class KinematicFinger

  Holds information about a finger from a KinematicHand.
- class KinematicHand

  Implements adaptive hand behaviour for a hand with fingers kinematically. This component is only responsible for the hands visuals.

4.2 AdaptiveHands.BendStates Namespace Reference

Classes

- class BendState

  A bend state that holds bent/unbent state for a hand.
- class BendStateSwapper

  A public class that allows 'bent' and 'unbent' hand states to be named, saved, and swapped at runtime or in the editor.

4.3 AdaptiveHands.Delegates Namespace Reference

Functions

- delegate void ActionRef< T > (ref T pItem)

  A simple delegate for events where an argument is passed by reference.
- delegate void ActionRef< VALUE_ONE, T > (VALUE_ONE pValueOne, ref T pItem)

  A simple delegate for events where an argument is passed by reference.

- delegate void ActionRef< VALUE_ONE, VALUE_TWO, T > (VALUE_ONE pValueOne, VALUE_TWO p↩ ValueTwo, ref T pItem)

  A simple delegate for events where an argument is passed by reference.
- delegate void ActionRef< VALUE_ONE, VALUE_TWO, VALUE_THREE, T > (VALUE_ONE pValueOne, VALUE_TWO pValueTwo, VALUE_THREE pValueThree, ref T pItem)

  A simple delegate for events where an argument is passed by reference.
- delegate void DoubleActionRef< T1, T2 > (ref T1 pItemA, ref T2 pItemB)

  A simple delegate for events where two arguments are passed by reference.
- delegate void DoubleActionRef< V1, T1, T2 > (V1 pValueA, ref T1 pItemA, ref T2 pItemB)

  A simple delegate for events where two arguments are passed by reference.
- delegate void DoubleActionRef< V1, V2, T1, T2 > (V1 pValueA, V2 pValueB, ref T1 pItemA, ref T2 pItemB)

  A simple delegate for events where two arguments are passed by reference.
- delegate void DoubleActionRef< V1, V2, V3, T1, T2 > (V1 pValueA, V2 pValueB, V3 pValueC, ref T1 pItemA, ref T2 pItemB)

  A simple delegate for events where two arguments are passed by reference.

4.3.1 Function Documentation

4.3.1.1 ActionRef< T >()

```
delegate void AdaptiveHands.Delegates.ActionRef< T > (
            ref T pItem )
```

A simple delegate for events where an argument is passed by reference.

Template Parameters

T	The type of the passed reference.

Parameters

pItem	The reference that was passed.

4.3.1.2 ActionRef< VALUE_ONE, T >()

```
delegate void AdaptiveHands.Delegates.ActionRef< VALUE_ONE, T > (
            VALUE_ONE pValueOne,
            ref T pItem )
```

A simple delegate for events where an argument is passed by reference.

Template Parameters

VALUE_ONE	The type of the fist passed value.
T	The type of the passed reference.

Parameters

pValueOne	The first value that was passed.
pItem	The reference that was passed.

4.3.1.3 ActionRef< VALUE_ONE, VALUE_TWO, T >()

```
delegate void AdaptiveHands.Delegates.ActionRef< VALUE_ONE, VALUE_TWO, T > (
             VALUE_ONE pValueOne,
             VALUE_TWO pValueTwo,
             ref T pItem )
```

A simple delegate for events where an argument is passed by reference.

Template Parameters

VALUE_ONE	The type of the first passed value.
VALUE_TWO	The type of the second passed value.
T	The type of the passed reference.

Parameters

pValueOne	The first value that was passed.
pValueTwo	The second value that was passed.
pItem	The reference that was passed.

4.3.1.4 ActionRef< VALUE_ONE, VALUE_TWO, VALUE_THREE, T >()

```
delegate void AdaptiveHands.Delegates.ActionRef< VALUE_ONE, VALUE_TWO, VALUE_THREE, T > (
             VALUE_ONE pValueOne,
             VALUE_TWO pValueTwo,
             VALUE_THREE pValueThree,
             ref T pItem )
```

A simple delegate for events where an argument is passed by reference.

Template Parameters

VALUE_ONE	The type of the first passed value.
VALUE_TWO	The type of the second passed value.
VALUE_THREE	The type of the third passed value.
T	The type of the passed reference.

Parameters

pValueOne	The first value that was passed.
pValueTwo	The second value that was passed.
pValueThree	The third value that was passed.
pItem	The reference that was passed.

4.3.1.5 DoubleActionRef< T1, T2 >()

```
delegate void AdaptiveHands.Delegates.DoubleActionRef< T1, T2 > (
             ref T1 pItemA,
             ref T2 pItemB )
```

A simple delegate for events where two arguments are passed by reference.

Template Parameters

T1	The type of the first passed reference.
T2	The type of the second passed reference.

Parameters

pItemA	The first reference that was passed.
pItemB	The second reference that was passed.

4.3.1.6 DoubleActionRef< V1, T1, T2 >()

```
delegate void AdaptiveHands.Delegates.DoubleActionRef< V1, T1, T2 > (
             V1 pValueA,
             ref T1 pItemA,
             ref T2 pItemB )
```

A simple delegate for events where two arguments are passed by reference.

Template Parameters

V1	The type of the first passed value.
T1	The type of the first passed reference.
T2	The type of the second passed reference.

Parameters

pValueA	The first value that was passed.
pItemA	The first reference that was passed.

Parameters

pItemB	The second reference that was passed.

4.3.1.7 DoubleActionRef< V1, V2, T1, T2 >()

```
delegate void AdaptiveHands.Delegates.DoubleActionRef< V1, V2, T1, T2 > (
             V1 pValueA,
             V2 pValueB,
             ref T1 pItemA,
             ref T2 pItemB )
```

A simple delegate for events where two arguments are passed by reference.

Template Parameters

V1	The type of the first passed value.
V2	The type of the second passed value.
T1	The type of the first passed reference.
T2	The type of the second passed reference.

Parameters

pValueA	The first value that was passed.
pValueB	The second value that was passed.
pItemA	The first reference that was passed.
pItemB	The second reference that was passed.

4.3.1.8 DoubleActionRef< V1, V2, V3, T1, T2 >()

```
delegate void AdaptiveHands.Delegates.DoubleActionRef< V1, V2, V3, T1, T2 > (
             V1 pValueA,
             V2 pValueB,
             V3 pValueC,
             ref T1 pItemA,
             ref T2 pItemB )
```

A simple delegate for events where two arguments are passed by reference.

Template Parameters

V1	The type of the first passed value.
V2	The type of the second passed value.
V3	The type of the third passed value.
T1	The type of the first passed reference.
T2	The type of the second passed reference.

Parameters

pValueA	The first value that was passed.
pValueB	The second value that was passed.
pValueC	The third value that was passed.
pItemA	The first reference that was passed.
pItemB	The second reference that was passed.

4.4 AdaptiveHands.Editor Namespace Reference

Classes

- class **AdaptiveHandsEditorSettings**

  A public static class that stores settings for Adaptive Hands.
- class AdaptiveHandsEditorSettingsWindow

  A window where adaptive hands settings can be modified.
- class BendStateSwapperEditor

  A custom inspector for the BendStateSwapper component.
- class **EditModeHandSimulator**

  A static class that allows a KinematicHand to be simulated in edit mode.
- class **EditorSymmetryUtility**

  A public static class that provides extra runtime methods relating to symmetry.
- class HandPoserEditor

  A custom inspector for the HandPoser component.
- class HandSymmetryToolWindow

  A window where adaptive hand components and settings can be copied from a symmetrical hand.
- class KinematicHandEditor

  A custom inspector for KinematicHands.
- class PoseSymmetryToolWindow

  A window where adaptive hand components and settings from HnadPoser or BendStateSwapper components can be copied from a symmetrical hand.
- class **UndoTracker**

  A public static class that tracks editor undo groups that can not be easily grouped.

4.5 AdaptiveHands.Editor.Animation Namespace Reference

Classes

- class **GenericAnimationExporter**

  A public static class that provides methods to export generic animations.
- class GenericAnimationExporterWindow

  A tool designed to make it easy to export generic hand animations.
- class **HumanoidAnimationExporter**

  A public static class that provides methods to export adaptive hand animations.
- class HumanoidAnimationExporterWindow

  A tool designed to make it easy to export humanoid hand animations.

4.6 AdaptiveHands.Events Namespace Reference

Classes

- class BendStateAreaUnityEvent

  Arg0: BendStateArea - The BendStateArea involved in the event. Arg1: BendStateSwapper - The BendStateSwapper involved in the event.
- class HandPoseAreaUnityEvent

  Arg0: HandPoseArea - The HandPoseArea involved in the event. Arg1: HandPoser - The HandPoser involved in the event.

4.7 AdaptiveHands.Poser Namespace Reference

Classes

- class HandPose

  A hand pose.
- class HandPoser

  A component that allows poses to be saved and loaded for a hand.

4.8 AdaptiveHands.Triggers Namespace Reference

Classes

- class BendStateArea

  A simple component that uses the 'OnTriggerEnter' and 'OnTriggerExit' callbacks to set a state on a BendState↩ Swapper that enters a trigger and clears it when it exits the trigger.
- class HandPoseArea

  A simple component that uses the 'OnTriggerEnter' and 'OnTriggerExit' callbacks to set a state on a HandPoser that enters a trigger and clears it when it exits the trigger.

4.9 AdaptiveHands.Utility Namespace Reference

Classes

- class **GizmoUtility**

  A public static class that provides extra runtime methods relating to gizmos.
- class **TransformUtility**

  A public static class that provides helper methods for working with Transforms.

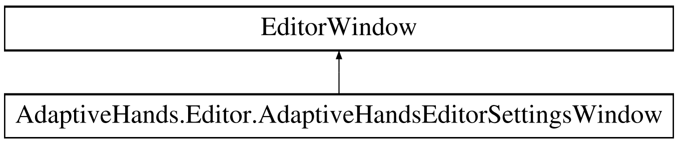
Chapter 5

Class Documentation

5.1 AdaptiveHands.Editor.AdaptiveHandsEditorSettingsWindow Class Reference

A window where adaptive hands settings can be modified.

Inheritance diagram for AdaptiveHands.Editor.AdaptiveHandsEditorSettingsWindow:



Public Member Functions

- void **ResetGlobalSettings** ()

  Resets and overwrites the AdaptiveHandsEditorSettings static classes' settings with the ones from this AdaptiveHandsEditorSettingsWindow.

Events

- static Action< AdaptiveHandsEditorSettingsWindow > **Initialized**

  A C# delegate event that is invoked when the AdaptiveHandsEditorSettingsWindow is intialized.

5.1.1 Detailed Description

A window where adaptive hands settings can be modified.

Author: Mathew Aloisio

The documentation for this class was generated from the following file:

- AdaptiveHandsEditorSettingsWindow.cs

5.2 AdaptiveHands.BendStates.BendState Class Reference

A bend state that holds bent/unbent state for a hand.

Classes

- class BoneEntry

Public Member Functions

- **BendState** (BendState pOther)
- BoneEntry[ ] CopyBendStateData ()

  Generates and returns a deep copy of the 'bendStateData' array for this BendState.

Public Attributes

- string **name**

  The name of the bend state.
- BoneEntry[ ] **bendStateData**

  The bend state data for the bend state.

5.2.1 Detailed Description

A bend state that holds bent/unbent state for a hand.

5.2.2 Member Function Documentation

5.2.2.1 CopyBendStateData()

```
BoneEntry[ ] AdaptiveHands.BendStates.BendState.CopyBendStateData ( )
```

Generates and returns a deep copy of the 'bendStateData' array for this BendState.

Returns

a deep copy of the 'bendStateData' array for this BendState.

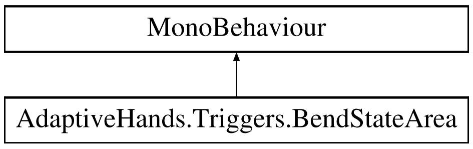
The documentation for this class was generated from the following file:

- BendState.cs

5.3 AdaptiveHands.Triggers.BendStateArea Class Reference

A simple component that uses the 'OnTriggerEnter' and 'OnTriggerExit' callbacks to set a state on a BendState↩ Swapper that enters a trigger and clears it when it exits the trigger.

Inheritance diagram for AdaptiveHands.Triggers.BendStateArea:



Public Attributes

- string **bendState**

  The name of the bend state to attempt to put the BendStateSwapper into when it enters the bend state area..
- bool **checkRigidbody**

  If not found on the triggering Collider
- bool **checkTriggerStay**

  An option that allows for triggers to be considered as
- bool **clearOnExit** = true

  Should the set bend state be cleared on exit from the area?
- BendStateAreaUnityEvent **SwapperEnteringArea**

  An event that is invoked just before a BendStateSwapper enters the area.
  \nArg
- BendStateAreaUnityEvent **SwapperEnteredArea**

  An event that is invoked whenever a BendStateSwapper enters the area.
  \nArg
- BendStateAreaUnityEvent **SwapperExitingArea**

  An event that is invoked just before a BendStateSwapper exits the area.
  \nArg
- BendStateAreaUnityEvent **SwapperExitedArea**

  An event that is invoked whenever a BendStateSwapper exits the area.
  \nArg

Events

- ActionRef< BendStateArea, BendStateSwapper, bool > **BlockBendStateSwapDelegate**

  A C# event delegate that provides the opportunity for external scripts to block the bend state area from posing under certain conditions. Arg0: BendStateArea - The BendStateArea triggering the bend state swapping. Arg1: Bend↩ StateSwapper - The BendStateSwapper being triggered by the area. Arg2: ref bool - If true the bend state swap is blocked, otherwise if false the bend state area will function normally.

5.3.1 Detailed Description

A simple component that uses the 'OnTriggerEnter' and 'OnTriggerExit' callbacks to set a state on a BendState↩ Swapper that enters a trigger and clears it when it exits the trigger.

Author: Mathew Aloisio

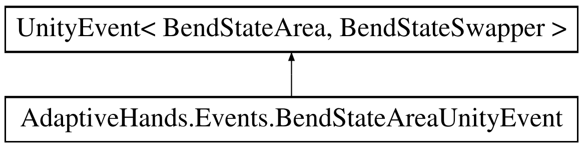
The documentation for this class was generated from the following file:

- BendStateArea.cs

5.4 AdaptiveHands.Events.BendStateAreaUnityEvent Class Reference

Arg0: BendStateArea - The BendStateArea involved in the event. Arg1: BendStateSwapper - The BendState↩ Swapper involved in the event.

Inheritance diagram for AdaptiveHands.Events.BendStateAreaUnityEvent:


UnityEvent< BendStateArea, BendStateSwapper >
AdaptiveHands.Events.BendStateAreaUnityEvent

5.4.1 Detailed Description

Arg0: BendStateArea - The BendStateArea involved in the event. Arg1: BendStateSwapper - The BendState↩ Swapper involved in the event.

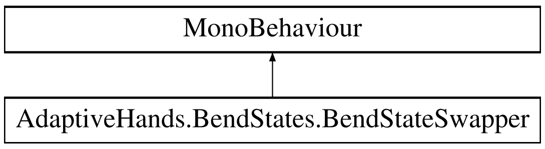
The documentation for this class was generated from the following file:

- BendStateAreaUnityEvent.cs

5.5 AdaptiveHands.BendStates.BendStateSwapper Class Reference

A public class that allows 'bent' and 'unbent' hand states to be named, saved, and swapped at runtime or in the editor.

Inheritance diagram for AdaptiveHands.BendStates.BendStateSwapper:



Public Member Functions

- void **SetBendStatesToCurrentState** ()

  Sets the bend states of the Hands finger bones to the current state.
- void SetBendStatesToState (BendState pState)

  Sets the bend states of the Hands finger bones to the given state, pState.
- void SetBendStatesToStateIndex (int pIndex)

  Sets the bend states of the Hands finger bones to the given state index.
- void SetState (string pStateName)

  Sets the poser to the state with the given name. If not found or pStateName is null the state is cleared.
- void SetStateByIndex (int pIndex)

  Sets the state by index.
- void **ClearState** ()

Clears the current state.

- void **SaveDefaultNoBendState** ()

  Saves/overwrites the default 'full bend' info state using the current settings for the relevant hand.
- void **SaveDefaultFullBendState** ()

  Saves/overwrites the default 'full bend' info state using the current settings for the relevant hand.
- void SaveNewState (string pStateName)

  Saves a new state with the given name and the default bend settings.
- void SaveNoBendState (string pStateName)

  Saves the current Hand 'full bend' state as a state with the given name pStateName. Overwrites existing entries.
- void SaveNoBendStateByIndex (int pIndex)

  Saves the current Hand 'full bend' state overwriting the state in the given index, pIndex. Overwrites existing entries.
- void SaveFullBendState (string pStateName)

  Saves the current Hand 'full bend' state as a state with the given name pStateName. Overwrites existing entries.
- void SaveFullBendStateByIndex (int pIndex)

  Saves the current Hand 'full bend' state overwriting the state in the given index, pIndex. Overwrites existing entries.
- void DeleteStateByName (string pStateName)

  Delets a state by name.
- void DeleteStateByIndex (int pIndex)

  Deletes a state by index.
- BendState GetStateByName (string pStateName)

  Returns the BendState with the given name, or null if not found. Note that 'defaultBendState' cannot be retrieved by name, only custom registered states can be retrieved using this method.
- int GetStateIndexByName (string pStateName)

  Returns the index of the state with the given name, or STATE_NONE (-1) if not found. Note that 'defaultBendState' cannot be retrieved by name, only custom registered states can be retrieved using this method.
- BendState GetStateByIndex (int pIndex)

  Returns the BendState at the given index.
- List< BendState.BoneEntry > GetCurrentBendStateData ()

  Generates and returns a List of BendState.BoneEntrys that contains the current finger bones full and full bend infos.
- BendState GetBendStateByName (string pBendStateName)

  Returns the BendState with the given name, or null if not found.
- int GetBendStateIndexByName (string pBendStateName)

  Returns the index of the pose with the given name, or STATE_NONE (-1) if not found.
- BendState GetBendStateByIndex (int pIndex)

  Retrieves the BendState in the given 'states' index, pIndex.
- void OverwriteBendStates (List< BendState > pStates)

  Overwrites the 'Bend State' List with the given one.

Public Attributes

- BendState **defaultBendState**

  Holds the default bend state for the relevant hand.

Static Public Attributes

- const int **STATE_NONE** = -1

  The value that represents no state index.

Properties

- int **CurrentStateIndex** `[get]`

  The current state index the poser is in, or STATE_NONE (-1) if not in any state.
- int **StateCount** `[get]`

  Returns the number of poses this component has registered.
- KinematicHand **Hand** `[get]`

  Returns the reference to the KinematicHand this poser belongs to.

5.5.1 Detailed Description

A public class that allows 'bent' and 'unbent' hand states to be named, saved, and swapped at runtime or in the editor.

Author: Mathew Aloisio

5.5.2 Member Function Documentation

5.5.2.1 DeleteStateByIndex()

```
void AdaptiveHands.BendStates.BendStateSwapper.DeleteStateByIndex (
             int pIndex )
```

Deletes a state by index.

Parameters

pIndex	

5.5.2.2 DeleteStateByName()

```
void AdaptiveHands.BendStates.BendStateSwapper.DeleteStateByName (
             string pStateName )
```

Delets a state by name.

Parameters

pStateName	

5.5.2.3 GetBendStateByIndex()

```
BendState AdaptiveHands.BendStates.BendStateSwapper.GetBendStateByIndex (
             int pIndex )
```

Retrieves the BendState in the given 'states' index, pIndex.

Parameters

pIndex	

Returns

the BendState in the given 'states' index, pIndex.

5.5.2.4 GetBendStateByName()

```
BendState AdaptiveHands.BendStates.BendStateSwapper.GetBendStateByName (
             string pBendStateName )
```

Returns the BendState with the given name, or null if not found.

Parameters

pBendStateName	

Returns

the BendState with the given name, or null if not found.

5.5.2.5 GetBendStateIndexByName()

```
int AdaptiveHands.BendStates.BendStateSwapper.GetBendStateIndexByName (
             string pBendStateName )
```

Returns the index of the pose with the given name, or STATE_NONE (-1) if not found.

Parameters

pBendStateName	

Returns

an int representing the index of the pose with the given name, or STATE_NONE (-1) if not found.

5.5.2.6 GetCurrentBendStateData()

```
List< BendState.BoneEntry > AdaptiveHands.BendStates.BendStateSwapper.GetCurrentBendStateData
( )
```

Generates and returns a List of BendState.BoneEntrys that contains the current finger bones full and full bend infos.

Returns

5.5.2.7 GetStateByIndex()

```
BendState AdaptiveHands.BendStates.BendStateSwapper.GetStateByIndex (
             int pIndex )
```

Returns the BendState at the given index.

Parameters

pIndex	

Returns

the BendState at the given index.

5.5.2.8 GetStateByName()

```
BendState AdaptiveHands.BendStates.BendStateSwapper.GetStateByName (
             string pStateName )
```

Returns the BendState with the given name, or null if not found. Note that 'defaultBendState' cannot be retrieved by name, only custom registered states can be retrieved using this method.

Parameters

pStateName	

Returns

the BendState with the given name, or null if not found.

5.5.2.9 GetStateIndexByName()

```
int AdaptiveHands.BendStates.BendStateSwapper.GetStateIndexByName (
            string pStateName )
```

Returns the index of the state with the given name, or STATE_NONE (-1) if not found. Note that 'defaultBendState' cannot be retrieved by name, only custom registered states can be retrieved using this method.

Parameters

pStateName	

Returns

an int representing the index of the state with the given name, or STATE_NONE (-1) if not found.

5.5.2.10 OverwriteBendStates()

```
void AdaptiveHands.BendStates.BendStateSwapper.OverwriteBendStates (
            List< BendState > pStates )
```

Overwrites the 'Bend State' List with the given one.

Parameters

pStates	A List of BendStates.

5.5.2.11 SaveFullBendState()

```
void AdaptiveHands.BendStates.BendStateSwapper.SaveFullBendState (
            string pStateName )
```

Saves the current Hand 'full bend' state as a state with the given name pStateName. Overwrites existing entries.

Parameters

pStateName	

5.5.2.12 SaveFullBendStateByIndex()

```
void AdaptiveHands.BendStates.BendStateSwapper.SaveFullBendStateByIndex (
            int pIndex )
```

Saves the current Hand 'full bend' state overwriting the state in the given index, pIndex. Overwrites existing entries.

NOTE: This method does not perform any error checking to ensure the state at pIndex is valid.

Parameters

pIndex	

5.5.2.13 SaveNewState()

```
void AdaptiveHands.BendStates.BendStateSwapper.SaveNewState (
             string pStateName )
```

Saves a new state with the given name and the default bend settings.

Parameters

pStateName	

5.5.2.14 SaveNoBendState()

```
void AdaptiveHands.BendStates.BendStateSwapper.SaveNoBendState (
             string pStateName )
```

Saves the current Hand 'full bend' state as a state with the given name pStateName. Overwrites existing entries.

Parameters

pStateName	

5.5.2.15 SaveNoBendStateByIndex()

```
void AdaptiveHands.BendStates.BendStateSwapper.SaveNoBendStateByIndex (
             int pIndex )
```

Saves the current Hand 'full bend' state overwriting the state in the given index, pIndex. Overwrites existing entries.

NOTE: This method does not perform any error checking to ensure the state at pIndex is valid.

Parameters

pIndex	

5.5.2.16 SetBendStatesToState()

```
void AdaptiveHands.BendStates.BendStateSwapper.SetBendStatesToState (
             BendState pState )
```

Sets the bend states of the Hands finger bones to the given state, pState.

Parameters

pState	The BendState to set the state to.

5.5.2.17 SetBendStatesToStateIndex()

```
void AdaptiveHands.BendStates.BendStateSwapper.SetBendStatesToStateIndex (
             int pIndex )
```

Sets the bend states of the Hands finger bones to the given state index.

Parameters

pIndex	The index of the BendState to set the state to.

5.5.2.18 SetState()

```
void AdaptiveHands.BendStates.BendStateSwapper.SetState (
             string pStateName )
```

Sets the poser to the state with the given name. If not found or pStateName is null the state is cleared.

Parameters

pStateName	

5.5.2.19 SetStateByIndex()

```
void AdaptiveHands.BendStates.BendStateSwapper.SetStateByIndex (
             int pIndex )
```

Sets the state by index.

Parameters

pIndex	

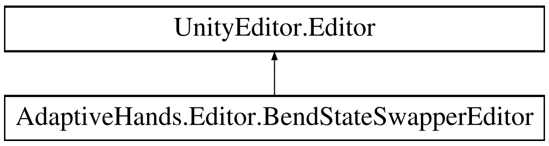
The documentation for this class was generated from the following file:

- BendStateSwapper.cs

5.6 AdaptiveHands.Editor.BendStateSwapperEditor Class Reference

A custom inspector for the BendStateSwapper component.

Inheritance diagram for AdaptiveHands.Editor.BendStateSwapperEditor:



Public Member Functions

- override void **OnInspectorGUI** ()

5.6.1 Detailed Description

A custom inspector for the BendStateSwapper component.

Author: Mathew Aloisio

The documentation for this class was generated from the following file:

- BendStateSwapperEditor.cs

5.7 AdaptiveHands.KinematicFinger.Bone Class Reference

A finger bone.

Public Member Functions

- **Bone** (Bone pOther)
- void **MoveToBend** ()

  Moves the finger bone to the current Bend position over time.
- void MoveToBend (float pBend)

  Moves the finger bone to the pBend position over time.
- void **SnapToBend** ()

  Snaps the finger bone to the current Bend position instantly.
- void SnapToBend (float pBend)

  Snaps the finger bone to the pBend position instantly.
- void **ZeroCurrentBend** ()

  Zeros the current bend for this finger bone and snaps to the zero'd bend.
- bool CorrectBend ()

  Corrects the Bend of the finger bone by checking all bend steps up til the current 'Bend' value.
- CapsulePoints GetCapsulePoints ()

  Returns a CapsulePoints instance that contains the 2 points of this fingers capsule collider. NOTE: If colliderLength == 0 the same point (center point of the sphere collider) is returned as both point1 and point2. WARNING: This method will throw an error if you invoke it while colliderTransform is null.
- bool CheckIfBlockedByCollision ()

  Checks if this finger bone is blocked by a collision and returns true if this finger bone is blocked by a collision, otherwise false.

Public Attributes

- Transform **bone**

  The Transform of the finger bone.
- float **targetBend**

  The target bend of the finger.
- BoneTransform **noBendInfo**

  Transform information about the finger while at no bend
- BoneTransform **fullBendInfo**

  Transform information about the finger while at full bend
- Transform **colliderTransform**
- Vector3 **colliderOffset** = Vector3.zero

  The collider offset in local space relative to colliderTransform.
- float **colliderRadius** = 0.007f

  The radius of the collider for this finger bone.
- float **colliderLength** = 0f

  The length from center
- Vector3 **colliderUp** = Vector3.up

  The up direction in local space for the collider for this bone. [The direction that points upwards from the bone
- Vector3 **colliderForward** = Vector3.forward

  The forward direction in local space for the collider for this bone. [The direction that points along the length of the bone.]

Properties

- float **Bend** = 0f `[get]`

  The actual current bend value for the finger bone. (0 - no bend | 1 - full bend)
- bool **CanBend** `[get]`

  Returns true if this bone could be bent last frame, otherwise false.
- bool **BendBlockedByCollision** `[get]`

  Returns true if this finger bone's bend is blocked by a collision, otherwise false.
- int **BoneIndex** `[get, set]`

  The index of this bone in it's KinematicFingers bones array.
- KinematicFinger **Finger** `[get, set]`

  A reference to the KinematicFinger this KinematicFinger.Bone belongs to.
- Vector3 **TargetLocalPosition** `[get, set]`

  The localPosition target this finger bone is moving towards.
- Quaternion **TargetLocalRotation** `[get, set]`

  The localRotation target this finger bone is moving towards.
- bool **DisableGizmos** `[get, set]`

  Allows gizmos for the finger bone to be forcibly disabled.
- Vector3 **WorldColliderUp** `[get]`

  Returns the world space 'up' direction based on the colliderUp direction given in colliderTransforms local space. WARNING: If colliderTransform is null this will cause an error.
- Vector3 **WorldColliderForward** `[get]`

  Returns the world space 'forward' direction based on the colliderForward direction given in colliderTransforms local space. WARNING: If colliderTransform is null this will cause an error.
- Vector3 **WorldColliderOffset** `[get]`

  Returns the world space offset for this bone based on the colliderOffset given in colliderTransforms local space. WARNING: If colliderTransform is null this will cause an error.
- Vector3 **OffsetColliderPosition** `[get]`

  Returns the world space position of the colliderTransform's position offset by colliderOffset in world spcae. WARNING: If colliderTransform is null this will cause an error.

Events

- DoubleActionRef< Bone, bool, bool > **OverrideCollisionCheckDelegate**

  A delegate that allows collision checking for the finger bone to be overridden by subscribers. Arg0↩ : KinematicFinger.Bone - the KinematicFinger.Bone whose collisions are being checked. Arg1: ref bool - a reference to the boolean that determine the resulting collision check result (true means collision blocking, false means no collision blocking.) Arg2: ref bool - a reference to a boolean that determines whether or not to override the collision check in the first place (making this true lets the collision check system know to use the value of 'arg0' as the collision check result.)

5.7.1 Detailed Description

A finger bone.

5.7.2 Member Function Documentation

5.7.2.1 CheckIfBlockedByCollision()

```
bool AdaptiveHands.KinematicFinger.Bone.CheckIfBlockedByCollision ( )
```

Checks if this finger bone is blocked by a collision and returns true if this finger bone is blocked by a collision, otherwise false.

Returns

true if this finger bone is blocked by a collision, otherwise false.

5.7.2.2 CorrectBend()

```
bool AdaptiveHands.KinematicFinger.Bone.CorrectBend ( )
```

Corrects the Bend of the finger bone by checking all bend steps up til the current 'Bend' value.

Returns

true if the bend value was adjusted due to a collision, otherwise false.

5.7.2.3 GetCapsulePoints()

```
CapsulePoints AdaptiveHands.KinematicFinger.Bone.GetCapsulePoints ( )
```

Returns a CapsulePoints instance that contains the 2 points of this fingers capsule collider. NOTE: If colliderLength == 0 the same point (center point of the sphere collider) is returned as both point1 and point2. WARNING: This method will throw an error if you invoke it while colliderTransform is null.

CALCULATIONS: point1 = colliderCenter point2 = colliderCenter + (WorldColliderForward ∗ colliderLength)

Returns

a CapsulePoints instance that contains the 2 points of this fingers capsule collider.

5.7.2.4 MoveToBend()

```
void AdaptiveHands.KinematicFinger.Bone.MoveToBend (
             float pBend )
```

Moves the finger bone to the pBend position over time.

Parameters

pBend	The bend factor. (0-1)

5.7.2.5 SnapToBend()

```
void AdaptiveHands.KinematicFinger.Bone.SnapToBend (
            float pBend )
```

Snaps the finger bone to the pBend position instantly.

Parameters

pBend	The bend factor. (0-1)

The documentation for this class was generated from the following file:

- KinematicFinger.cs

5.8 AdaptiveHands.BendStates.BendState.BoneEntry Class Reference

Public Attributes

- KinematicFinger **finger**

  A reference to the KinematicFinger the bone belongs to.
- int **index**

  The index of the bone.
- KinematicFinger.BoneTransform **noBendInfo**

  Transform information about the finger bone while at no bend
- KinematicFinger.BoneTransform **fullBendInfo**

  Transform information about the finger bone while at full bend

The documentation for this class was generated from the following file:

- BendState.cs

5.9 AdaptiveHands.Poser.HandPose.BoneEntry Class Reference

Public Attributes

- KinematicFinger **finger**

  A reference to the KinematicFinger the bone belongs to.
- int **index**

  The index of the bone.
- float **bend**

  The bend value of the bone.

The documentation for this class was generated from the following file:

- HandPose.cs

5.10 AdaptiveHands.KinematicFinger.BoneTransform Struct Reference

Public Attributes

- Vector3 **position**

  The position of the bone.
- Quaternion **rotation**

  The rotation of the bone.

The documentation for this struct was generated from the following file:

- KinematicFinger.cs

5.11 AdaptiveHands.KinematicFinger.CapsulePoints Struct Reference

Public Attributes

- Vector3 **point1**

  The first point of the capsule (the center of the first sphere of the capsule).
- Vector3 **point2**

  The second point of the capsule (the center of the second sphere of the capsule).

The documentation for this struct was generated from the following file:

- KinematicFinger.cs

5.12 AdaptiveHands.Triggers.HandPoseArea.Entry Class Reference

Public Attributes

- HandPoser **poser**

  A reference to the HandPoser.
- HandPose **cachedPose**

  A reference to the cached HandPose for the hand, otherwise null.

The documentation for this class was generated from the following file:

- HandPoseArea.cs

5.13 AdaptiveHands.Editor.AdaptiveHandsEditorSettings.Export↩ AnimationSettings Class Reference

Public Attributes

- string **exportPath** = "Assets/"

  The path to export animation clips to.

The documentation for this class was generated from the following file:

- AdaptiveHandsEditorSettings.cs

5.14 AdaptiveHands.Editor.KinematicHandEditor.FingerBoneHandle Struct Reference

Public Attributes

- KinematicFinger.Bone **bone**

  A reference to the KinematicFinger.Bone the handle is for.
- PrimitiveBoundsHandle **handle**

  A reference to the the resizable bounds handle.

The documentation for this struct was generated from the following file:

- KinematicHandEditor.cs

5.15 AdaptiveHands.FingerBoneTransforms Class Reference

A class that holds Transform references to all possible bones in a finger.

Public Attributes

- Transform **proximal**

  The proximal bone of the finger.
- Transform **intermediate**

  The intermediate bone of the finger.
- Transform **distal**

  The distal bone of the finger.

5.15.1 Detailed Description

A class that holds Transform references to all possible bones in a finger.

Author: Mathew Aloisio

The documentation for this class was generated from the following file:

- FingerBoneTransforms.cs

5.16 AdaptiveHands.Editor.HandSymmetryToolWindow.FlipAxes Class Reference

Public Attributes

- bool **x**
  Should the x axis be flipped?
- bool **y**
  Should the y axis be flipped?
- bool **z**
  Should the z axis be flipped?

The documentation for this class was generated from the following file:

- HandSymmetryToolWindow.cs

5.17 AdaptiveHands.Editor.PoseSymmetryToolWindow.FlipAxes Class Reference

Public Attributes

- bool **x**
  Should the x axis be flipped?
- bool **y**
  Should the y axis be flipped?
- bool **z**
  Should the z axis be flipped?

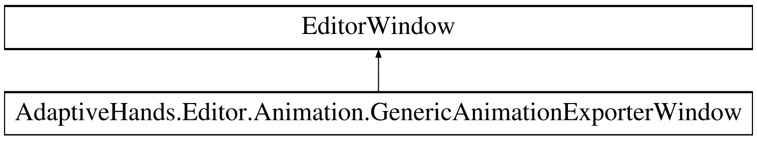
The documentation for this class was generated from the following file:

- PoseSymmetryToolWindow.cs

5.18 AdaptiveHands.Editor.Animation.GenericAnimationExporter↩ Window Class Reference

A tool designed to make it easy to export generic hand animations.

Inheritance diagram for AdaptiveHands.Editor.Animation.GenericAnimationExporterWindow:


EditorWindow
AdaptiveHands.Editor.Animation.GenericAnimationExporterWindow

Static Public Member Functions

- static void **Open** ()

Events

- static Action< GenericAnimationExporterWindow > **Initialized**

  An event that is invoked when the GenericAnimationExporterWindow is intialized.

5.18.1 Detailed Description

A tool designed to make it easy to export generic hand animations.

Author: Mathew Aloisio

The documentation for this class was generated from the following file:

- GenericAnimationExporterWindow.cs

5.19 AdaptiveHands.HandBoneTransforms Class Reference

A class that holds Transform references to all possible bones in a hand.

Public Attributes

- FingerBoneTransforms **thumb**

  The bones that make up a human avatar thumb.
- FingerBoneTransforms **index**

  The bones that make up a human avatar index finger.
- FingerBoneTransforms **middle**

  The bones that make up a human avatar middle finger.
- FingerBoneTransforms **ring**

  The bones that make up a human avatar ring finger.
- FingerBoneTransforms **pinky**

  The bones that make up a human avatar pinky finger.

5.19.1 Detailed Description

A class that holds Transform references to all possible bones in a hand.

Author: Mathew Aloisio

The documentation for this class was generated from the following file:

- HandBoneTransforms.cs

5.20 AdaptiveHands.Editor.AdaptiveHandsEditorSettings.Hand↩ DiagramSettings Class Reference

Public Attributes

- Texture **diagram**
  The color to render hand bone collider gizmos with.
- Color **distalColor** = new Color(.976f, .745f, .631f, 1f)
  The color of the distal bones in the hand bone diagram.
- Color **intermediateColor** = new Color(.506f, .729f, .878f, 1f)
  The color of the intermediate bones in the hand bone diagram.
- Color **proximalColor** = new Color(.808f, .906f, .678f, 1f)
  The color of the proximal bones in the hand bone diagram.

The documentation for this class was generated from the following file:

- AdaptiveHandsEditorSettings.cs

5.21 AdaptiveHands.Editor.AdaptiveHandsEditorSettings.Handle↩ Settings Class Reference

Public Attributes

- Color **colliderColor** = Color.white
  The color to use for collider handles.

The documentation for this class was generated from the following file:

- AdaptiveHandsEditorSettings.cs

5.22 AdaptiveHands.Poser.HandPose Class Reference

A hand pose.

Classes

- class BoneEntry

Public Member Functions

- **HandPose** (HandPose pOther)
- BoneEntry[ ] CopyHandPoseData ()

    Generates and returns a deep copy of the 'bendData' array for this HandPose.

Public Attributes

- string **name**

    The name of the hand pose.
- BoneEntry[ ] **bendData**

    The bend data for the hand pose.

5.22.1 Detailed Description

A hand pose.

5.22.2 Member Function Documentation

5.22.2.1 CopyHandPoseData()

```
BoneEntry[ ] AdaptiveHands.Poser.HandPose.CopyHandPoseData ( )
```

Generates and returns a deep copy of the 'bendData' array for this HandPose.

Returns

a deep copy of the 'bendData' array for this HandPose.

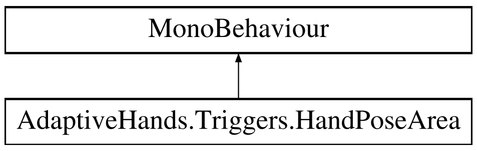
The documentation for this class was generated from the following file:

- HandPose.cs

5.23 AdaptiveHands.Triggers.HandPoseArea Class Reference

A simple component that uses the 'OnTriggerEnter' and 'OnTriggerExit' callbacks to set a state on a HandPoser that enters a trigger and clears it when it exits the trigger.

Inheritance diagram for AdaptiveHands.Triggers.HandPoseArea:



Classes

- class Entry

Public Types

- enum **BendExitMode**

Public Member Functions

- bool IsPoserInArea (HandPoser pPoser)

  Returns true if pPoser is in the area, otherwise false.

Public Attributes

- string **poseName**

  The name of the poseto attempt to put the HandPoser into when it enters the bend state area..
- bool **checkRigidbody**

  If not found on the triggering Collider
- bool **checkTriggerStay**

  An option that allows for triggers to be considered as
- bool **clearOnExit** = true

  Should the set pose be cleared on exit from the area?
- BendExitMode **bendExitMode** = BendExitMode.Restore

  Should the finger bend for the hand be zero
- HandPoseAreaUnityEvent **PoserEnteringArea**

  An event that is invoked just before a HandPoser enters the area.
  \nArg
- HandPoseAreaUnityEvent **PoserEnteredArea**

  An event that is invoked whenever a HandPoser enters the area.
  \nArg
- HandPoseAreaUnityEvent **PoserExitingArea**

  An event that is invoked just before a HandPoser exits the area.
  \nArg
- HandPoseAreaUnityEvent **PoserExitedArea**

  An event that is invoked whenever a HandPoser exits the area.
  \nArg

Events

- ActionRef< HandPoseArea, HandPoser, bool > **BlockHandPoseDelegate**

  A C# event delegate that provides the opportunity for external scripts to block the hand pose area from posing under certain conditions. Arg0: HandPoseArea - The HandPoseArea doing the posing. Arg1: HandPoser - The HandPoser being posed. Arg2: ref bool - If true the hand pose area is blocked, otherwise if false the pose area will function normally.

5.23.1 Detailed Description

A simple component that uses the 'OnTriggerEnter' and 'OnTriggerExit' callbacks to set a state on a HandPoser that enters a trigger and clears it when it exits the trigger.

Author: Mathew Aloisio

5.23.2 Member Function Documentation

5.23.2.1 IsPoserInArea()

```
bool AdaptiveHands.Triggers.HandPoseArea.IsPoserInArea (
            HandPoser pPoser )
```

Returns true if pPoser is in the area, otherwise false.

Parameters

pPoser	

Returns

true if pPoser is in the area, otherwise false.

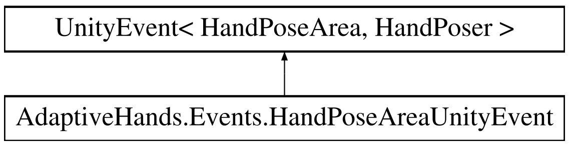
The documentation for this class was generated from the following file:

- HandPoseArea.cs

5.24 AdaptiveHands.Events.HandPoseAreaUnityEvent Class Reference

Arg0: HandPoseArea - The HandPoseArea involved in the event. Arg1: HandPoser - The HandPoser involved in the event.

Inheritance diagram for AdaptiveHands.Events.HandPoseAreaUnityEvent:


UnityEvent< HandPoseArea, HandPoser >
AdaptiveHands.Events.HandPoseAreaUnityEvent

5.24.1 Detailed Description

Arg0: HandPoseArea - The HandPoseArea involved in the event. Arg1: HandPoser - The HandPoser involved in the event.

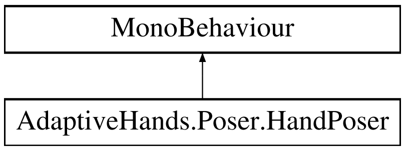
The documentation for this class was generated from the following file:

- HandPoseAreaUnityEvent.cs

5.25 AdaptiveHands.Poser.HandPoser Class Reference

A component that allows poses to be saved and loaded for a hand.

Inheritance diagram for AdaptiveHands.Poser.HandPoser:



Public Member Functions

- void **TrySetBendTargetsToCurrentPose** ()

  Invokes SetBendTargetsToCurrentPose() but only if the 'BlockPoseDelegate' does not override it.
- void **SetBendTargetsToCurrentPose** ()

  Sets the bend targets of the Hands finger bones to the current pose.
- void SetBendTargetsToPose (HandPose pPose)

  Sets the bend targets of the Hands finger bones to the given pose, pPose.
- void SetBendTargetsToPoseIndex (int pIndex)

  Sets the bend targets of the Hands finger bones to the given pose index.
- void SetPose (string pPoseName)

  Sets the poser to the pose with the given name. If not found or pPoseName is null the pose is cleared.
- void SetPoseByIndex (int pIndex)

  Sets the pose by index.
- void **ClearPose** ()

  Clears the current pose.
- void SavePose (string pPoseName)

  Saves the current Hand state as a pose with the given name pPoseName. Overwrites existing entries.
- void SavePoseByIndex (int pIndex)

  Saves the current Hand state in the pose at the given index. (Only usable for overwriting existing poses.)
- void DeletePoseByName (string pPoseName)

  Delets a pose by name.
- void DeletePoseByIndex (int pIndex)

  Deletes a pose by index.
- HandPose GetPoseByName (string pPoseName)

  Returns the HandPose with the given name, or null if not found.
- int GetPoseIndexByName (string pPoseName)

  Returns the index of the pose with the given name, or POSE_NONE (-1) if not found.
- HandPose GetCurrentHandPose ()

  Returns the current hand state as a HandPose or null if failed to generate.
- HandPose GetPoseByIndex (int pIndex)

  Returns the HandPose at the given index.
- void OverwritePoses (List< HandPose > pPoses)

  Overwrites the poses in this components 'poses' list with the given poses, pPoses.

Public Attributes

- ActionRef< HandPoser, bool > **BlockPoseDelegate**

  a C# delegate event that provides a reference to a boolean that allows you to specify whether or not a pose should be set. Arg0: HandPoser - The poser trying to set a hand pose. Arg1: ref bool - Should the pose be blocked? If true blocks posing, otherwise has no effect if false.

Static Public Attributes

- const int **POSE_NONE** = -1

  The value that represents no pose index.

Properties

- int **CurrentPoseIndex** `[get]`

  The current pose index the poser is in, or POSE_NONE (-1) if not in any pose.
- int **PoseCount** `[get]`

  Returns the number of poses this component has registered.
- KinematicHand **Hand** `[get]`

  Returns the reference to the KinematicHand this poser belongs to.

5.25.1 Detailed Description

A component that allows poses to be saved and loaded for a hand.

Author: Mathew Aloisio

5.25.2 Member Function Documentation

5.25.2.1 DeletePoseByIndex()

```
void AdaptiveHands.Poser.HandPoser.DeletePoseByIndex (
             int pIndex )
```

Deletes a pose by index.

Parameters

pIndex	

5.25.2.2 DeletePoseByName()

```
void AdaptiveHands.Poser.HandPoser.DeletePoseByName (
             string pPoseName )
```

Delets a pose by name.

Parameters

pPoseName	

5.25.2.3 GetCurrentHandPose()

```
HandPose AdaptiveHands.Poser.HandPoser.GetCurrentHandPose ( )
```

Returns the current hand state as a HandPose or null if failed to generate.

Returns

the current hand state as a HandPose or null if failed to generate.

5.25.2.4 GetPoseByIndex()

```
HandPose AdaptiveHands.Poser.HandPoser.GetPoseByIndex (
             int pIndex )
```

Returns the HandPose at the given index.

Parameters

pIndex	

Returns

the HandPose at the given index.

5.25.2.5 GetPoseByName()

```
HandPose AdaptiveHands.Poser.HandPoser.GetPoseByName (
             string pPoseName )
```

Returns the HandPose with the given name, or null if not found.

Parameters

pPoseName	

Returns

the HandPose with the given name, or null if not found.

5.25.2.6 GetPoseIndexByName()

```
int AdaptiveHands.Poser.HandPoser.GetPoseIndexByName (
             string pPoseName )
```

Returns the index of the pose with the given name, or POSE_NONE (-1) if not found.

Parameters

pPoseName	

Returns

an int representing the index of the pose with the given name, or POSE_NONE (-1) if not found.

5.25.2.7 OverwritePoses()

```
void AdaptiveHands.Poser.HandPoser.OverwritePoses (
             List< HandPose > pPoses )
```

Overwrites the poses in this components 'poses' list with the given poses, pPoses.

Parameters

pPoses	

5.25.2.8 SavePose()

```
void AdaptiveHands.Poser.HandPoser.SavePose (
             string pPoseName )
```

Saves the current Hand state as a pose with the given name pPoseName. Overwrites existing entries.

Parameters

pPoseName	

5.25.2.9 SavePoseByIndex()

```
void AdaptiveHands.Poser.HandPoser.SavePoseByIndex (
             int pIndex )
```

Saves the current Hand state in the pose at the given index. (Only usable for overwriting existing poses.)

Parameters

pIndex	

5.25.2.10 SetBendTargetsToPose()

```
void AdaptiveHands.Poser.HandPoser.SetBendTargetsToPose (
             HandPose pPose )
```

Sets the bend targets of the Hands finger bones to the given pose, pPose.

Parameters

pPose	The HandPose to set the pose to.

5.25.2.11 SetBendTargetsToPoseIndex()

```
void AdaptiveHands.Poser.HandPoser.SetBendTargetsToPoseIndex (
             int pIndex )
```

Sets the bend targets of the Hands finger bones to the given pose index.

Parameters

pIndex	The index of the HandPose to set the pose to.

5.25.2.12 SetPose()

```
void AdaptiveHands.Poser.HandPoser.SetPose (
            string pPoseName )
```

Sets the poser to the pose with the given name. If not found or pPoseName is null the pose is cleared.

Parameters

pPoseName	

5.25.2.13 SetPoseByIndex()

```
void AdaptiveHands.Poser.HandPoser.SetPoseByIndex (
            int pIndex )
```

Sets the pose by index.

Parameters

pIndex	

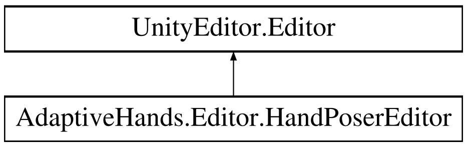
The documentation for this class was generated from the following file:

- HandPoser.cs

5.26 AdaptiveHands.Editor.HandPoserEditor Class Reference

A custom inspector for the HandPoser component.

Inheritance diagram for AdaptiveHands.Editor.HandPoserEditor:



Public Member Functions

- override void **OnInspectorGUI** ()

5.26.1 Detailed Description

A custom inspector for the HandPoser component.

Author: Mathew Aloisio

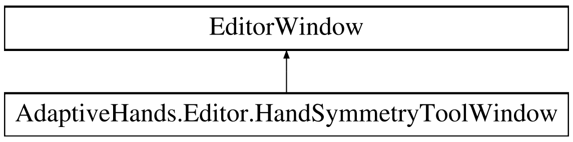
The documentation for this class was generated from the following file:

- HandPoserEditor.cs

5.27 AdaptiveHands.Editor.HandSymmetryToolWindow Class Reference

A window where adaptive hand components and settings can be copied from a symmetrical hand.

Inheritance diagram for AdaptiveHands.Editor.HandSymmetryToolWindow:



$$\begin{tikzcd}
\text{EditorWindow} \\
\text{AdaptiveHands.Editor.HandSymmetryToolWindow} \arrow[u]
\end{tikzcd}$$

Classes

- class FlipAxes

Public Member Functions

- void SymmetrizeHands (KinematicHand pSourceHand, KinematicHand pDestinationHand, bool pFlip↩ColliderUp, bool pFlipColliderForward)

    Symmetrizes pSourceHand and pDestinationHand by copying all relevant data from pSourceHand to pDestination↩ Hand.

Public Attributes

- KinematicHand **sourceHand**

    The source KinematicHand to copy from.
- KinematicHand **destinationHand**

    The destination KinematicHand to copy to.
- bool **flipColliderUp** = true

    Should the collider up direction be flipped for the destination hands finger bones?
- bool **flipColliderForward** = true

    Should the collider forward direction be flipped for the destination hands finger bones?
- FlipAxes **flipColliderAxes** = new FlipAxes() { x = true, y = false, z = true }

    What collider axes should be flipped on the destination hands fingers?
- FlipAxes **flipBendAxes** = new FlipAxes() { x = true, y = true, z = true }

    What bend info axes should be flipped on the destination hands fingers?
- FlipAxes **flipBendAngleAxes** = new FlipAxes() { x = false, y = false, z = false }

    What bend info angle axes should be flipped on the dstination hands fingers?

Events

- static Action< HandSymmetryToolWindow > **Initialized**

  A C# delegate event that is invoked when the HandSymmetryToolWindow is intialized.

5.27.1 Detailed Description

A window where adaptive hand components and settings can be copied from a symmetrical hand.

Author: Mathew Aloisio

5.27.2 Member Function Documentation

5.27.2.1 SymmetrizeHands()

```
void AdaptiveHands.Editor.HandSymmetryToolWindow.SymmetrizeHands (
            KinematicHand pSourceHand,
            KinematicHand pDestinationHand,
            bool pFlipColliderUp,
            bool pFlipColliderForward )
```

Symmetrizes pSourceHand and pDestinationHand by copying all relevant data from pSourceHand to pDestination↩ Hand.

Parameters

pSourceHand	
pDestinationHand	
pFlipColliderUp	Should the finger bones collider up direction be flipped?
pFlipColliderForward	Should the finger bones collider forward direction be flipped?

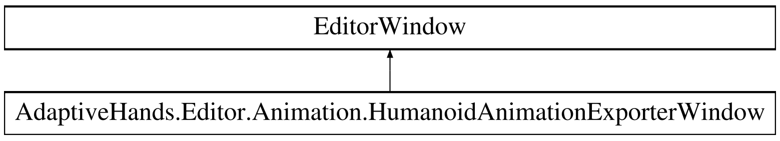
The documentation for this class was generated from the following file:

- HandSymmetryToolWindow.cs

5.28 AdaptiveHands.Editor.Animation.HumanoidAnimationExporter↩ Window Class Reference

A tool designed to make it easy to export humanoid hand animations.

Inheritance diagram for AdaptiveHands.Editor.Animation.HumanoidAnimationExporterWindow:


EditorWindow
AdaptiveHands.Editor.Animation.HumanoidAnimationExporterWindow

Static Public Member Functions

- static void **Open** ()

Events

- static Action< HumanoidAnimationExporterWindow > **Initialized**

  An event that is invoked when the HandAnimationExporterWindow is intialized.

5.28.1 Detailed Description

A tool designed to make it easy to export humanoid hand animations.

Author: Mathew Aloisio

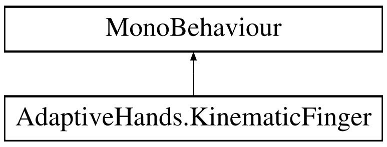
The documentation for this class was generated from the following file:

- HumanoidAnimationExporterWindow.cs

5.29 AdaptiveHands.KinematicFinger Class Reference

Holds information about a finger from a KinematicHand.

Inheritance diagram for AdaptiveHands.KinematicFinger:



Classes

- class Bone

  A finger bone.
- struct BoneTransform
- struct CapsulePoints

Public Member Functions

- void **UpdateBend** ()

  Updates the bend state for all bones in the finger.
- void **CalculateAverageBend** ()

  (Re)calculates 'AverageBend' using the average of all 'Bone.Bend' values for all finger bones in this finger.
- void **ZeroCurrentBend** ()

  Zeros and snaps to the zero'd currrent bend for all bones in the finger.
- void SetUnbendObstructed (bool pUnbendObstructed)

  A public method that allows the 'unbendObstructed' field of the KinematicFinger to be set. Useful for use with Unity editor events.
- void **ValidateBones** ()

  For editor-only purposes. Ensures all child finger bones know their finger reference.

Public Attributes

- bool **unbendObstructed** = true

  Automatically unbend an obstructed finger?
- LayerMask **ignoreBendLayers**

  A LayerMask of layers that the finger should ignore while bending.
- int **bendSteps** = 24

  The number of
- Bone[ ] **bones**

  An array of KinematicFinger.Bones that make up the finger.
- float **fingerMoveRate** = 0.1f

  The rate at which this hands fingers move to their target position at in units per second.
- float **fingerRotateRate** = 360f

  The rate at which this hands fingers rotates to their target rotation at in degrees per second.

Properties

- float **AverageBend** `[get]`

  Returns the average 'Bend' value for all bones in this finger combined as of the last call to 'UpdateBend()'.
- static Color **BoneGizmoColor** = new Color(1, 0, 0, 0.25f) `[get, set]`

  The Color used to render finger bone gizmos.

5.29.1 Detailed Description

Holds information about a finger from a KinematicHand.

Author: Mathew Aloisio

5.29.2 Member Function Documentation

5.29.2.1 SetUnbendObstructed()

```
void AdaptiveHands.KinematicFinger.SetUnbendObstructed (
            bool pUnbendObstructed )
```

A public method that allows the 'unbendObstructed' field of the KinematicFinger to be set. Useful for use with Unity editor events.

Parameters

pUnbendObstructed	

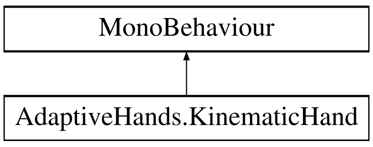
The documentation for this class was generated from the following file:

- KinematicFinger.cs

5.30 AdaptiveHands.KinematicHand Class Reference

Implements adaptive hand behaviour for a hand with fingers kinematically. This component is only responsible for the hands visuals.

Inheritance diagram for AdaptiveHands.KinematicHand:



Public Types

- enum **UpdateMode**

Public Member Functions

- void **UpdateHandFullBend** ()

  Forces the hand to simulate an update enough times to complete a full bend.
- void **ZeroAllFingerCurrentBend** ()

  Immediately zeroes the current finger bend (and snaps to it) for all fingers in the hand.
- void **SetAllFingerBendToCurrent** ()

  Overrides the target bend value for all bones in all fingers of the hand with their current bend value.
- void SetAllFingerBend (float pClosedness)

  Sets all finger bone bend targets to pClosedness.
- void **ZeroFingerCurrentBend** (KinematicFinger pFinger)

  Immediately zeroes the current finger bend (and snaps to it) for all fingers in the hand.
- void SetFingerBendToCurrent (KinematicFinger pFinger)

  Overrides the target bend value for all bones in pFinger with their current bend value.
- void SetFingerBend (KinematicFinger pFinger, float pClosedness)

  Sets all finger bone bend targets to pClosedness.
- void **ZeroFingerBoneCurrentBend** (KinematicFinger.Bone pBone)

  Immediately zeroes the current finger bone bend (and snaps to it) for the specified finger bone, pBone.
- void SetFingerBoneBendToCurrent (KinematicFinger.Bone pBone)

  Overrides the target bend value for pBone with the current bend value.
- void SetFingerBoneBend (KinematicFinger.Bone pBone, float pClosedness)

  Sets all finger bone bend targets to pClosedness.
- void SetUnbendObstructed (bool pUnbendObstructed)

  Sets the 'unbendObstructed' field for all KinematicFingers that make up the hand at the same time.
- void **UpdateHand** ()

  Updates the hand, including the bend state for all fingers that are part of the hand.

Public Attributes

- UpdateMode **updateMode** = UpdateMode.Update

  Controls when the hand is updated.
  \nManual
- KinematicFinger[ ] **fingers**

  An array of KinematicFinger components that define the fingers that make up the hand.

Properties

- float **AverageFingerBend** `[get]`

  The average actual finger bend value for all finger bones that make up this hand as calculated in the last call to 'UpdateHand()'. Non-enabled finger components are ignored.

5.30.1 Detailed Description

Implements adaptive hand behaviour for a hand with fingers kinematically. This component is only responsible for the hands visuals.

Author: Mathew Aloisio

5.30.2 Member Function Documentation

5.30.2.1 SetAllFingerBend()

```
void AdaptiveHands.KinematicHand.SetAllFingerBend (
            float pClosedness )
```

Sets all finger bone bend targets to pClosedness.

Parameters

pClosedness	The close (bend) factor for the hand's fingers. (0-1)

5.30.2.2 SetFingerBend()

```
void AdaptiveHands.KinematicHand.SetFingerBend (
            KinematicFinger pFinger,
            float pClosedness )
```

Sets all finger bone bend targets to pClosedness.

Parameters

pFinger	The KinematicFinger to set the bend values for.
pClosedness	The close (bend) factor for the finger. (0-1)

5.30.2.3 SetFingerBendToCurrent()

```
void AdaptiveHands.KinematicHand.SetFingerBendToCurrent (
            KinematicFinger pFinger )
```

Overrides the target bend value for all bones in pFinger with their current bend value.

Parameters

pFinger	

5.30.2.4 SetFingerBoneBend()

```
void AdaptiveHands.KinematicHand.SetFingerBoneBend (
            KinematicFinger.Bone pBone,
            float pClosedness )
```

Sets all finger bone bend targets to pClosedness.

Parameters

pBone	The KinematicFinger.Bone to set the bend value for.
pClosedness	The close (bend) factor for the finger bone. (0-1)

5.30.2.5 SetFingerBoneBendToCurrent()

```
void AdaptiveHands.KinematicHand.SetFingerBoneBendToCurrent (
            KinematicFinger.Bone pBone )
```

Overrides the target bend value for pBone with the current bend value.

Parameters

pBone	

5.30.2.6 SetUnbendObstructed()

```
void AdaptiveHands.KinematicHand.SetUnbendObstructed (
            bool pUnbendObstructed )
```

Sets the 'unbendObstructed' field for all KinematicFingers that make up the hand at the same time.

Parameters

pUnbendObstructed	

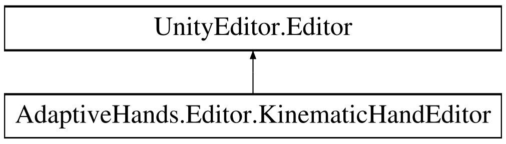
The documentation for this class was generated from the following file:

- KinematicHand.cs

5.31 AdaptiveHands.Editor.KinematicHandEditor Class Reference

A custom inspector for KinematicHands.

Inheritance diagram for AdaptiveHands.Editor.KinematicHandEditor:



Classes

- struct FingerBoneHandle

Public Member Functions

- override void **OnInspectorGUI** ()
- void SetAllFingerBend (KinematicHand pHand, float pBend)

  Invokes pHand.SetAllFingerBend(pBend) and performs other editor-related tasks.
- void SetFingerBend (KinematicHand pHand, KinematicFinger pFinger, float pBend)

  Invokes pHand.SetFingerBend(pFinger, pBend) and performs other editor-related tasks.
- void StoreOpenedHandPosition (KinematicHand pHand)

  Stores the finger positions and rotations for the opened hand position.
- void StoreClosedHandPosition (KinematicHand pHand)

  Stores the finger positions and rotations for the closed hand position.
- void StartEditingFingerBone (KinematicFinger.Bone pBone)

  Starts editing the specified finger bone using handles.
- void **StopEditingFingerBone** ()

  Stops editing any finger bone and cleans up the handles.
- void DrawPoseEditor (bool pShowFingerBoneEdit=true)

  Draws the pose editor for the KinematicHand. This may only be invoked in editor GUI methods.
- void **DrawFingerBoneColliderHandle** ()

  Draws the modifiable handle for the finger bone collider. NOTE: This must be called inside of an 'OnSceneGUI' method or any method where Camera.current is non-null.

5.31.1 Detailed Description

A custom inspector for KinematicHands.

Author: Mathew Aloisio

5.31.2 Member Function Documentation

5.31.2.1 DrawPoseEditor()

```
void AdaptiveHands.Editor.KinematicHandEditor.DrawPoseEditor (
            bool pShowFingerBoneEdit = true )
```

Draws the pose editor for the KinematicHand. This may only be invoked in editor GUI methods.

Parameters

pShowFingerBoneEdit	Should the bone collider edit buttons be drawn? (True - yes | False - no)

5.31.2.2 SetAllFingerBend()

```
void AdaptiveHands.Editor.KinematicHandEditor.SetAllFingerBend (
            KinematicHand pHand,
            float pBend )
```

Invokes pHand.SetAllFingerBend(pBend) and performs other editor-related tasks.

Parameters

pHand	
pBend	

5.31.2.3 SetFingerBend()

```
void AdaptiveHands.Editor.KinematicHandEditor.SetFingerBend (
            KinematicHand pHand,
            KinematicFinger pFinger,
            float pBend )
```

Invokes pHand.SetFingerBend(pFinger, pBend) and performs other editor-related tasks.

Parameters

pHand	
pFinger	
pBend	

5.31.2.4 StartEditingFingerBone()

```
void AdaptiveHands.Editor.KinematicHandEditor.StartEditingFingerBone (
            KinematicFinger.Bone pBone )
```

Starts editing the specified finger bone using handles.

Parameters

pBone	

5.31.2.5 StoreClosedHandPosition()

```
void AdaptiveHands.Editor.KinematicHandEditor.StoreClosedHandPosition (
            KinematicHand pHand )
```

Stores the finger positions and rotations for the closed hand position.

Parameters

pHand	

5.31.2.6 StoreOpenedHandPosition()

```
void AdaptiveHands.Editor.KinematicHandEditor.StoreOpenedHandPosition (
            KinematicHand pHand )
```

Stores the finger positions and rotations for the opened hand position.

Parameters

pHand	

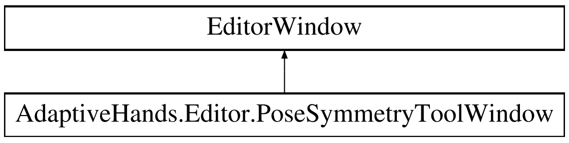
The documentation for this class was generated from the following file:

- KinematicHandEditor.cs

5.32 AdaptiveHands.Editor.PoseSymmetryToolWindow Class Reference

A window where adaptive hand components and settings from HnadPoser or BendStateSwapper components can be copied from a symmetrical hand.

Inheritance diagram for AdaptiveHands.Editor.PoseSymmetryToolWindow:



Classes

- class FlipAxes

Public Member Functions

- void SymmetrizeHandPosers (HandPoser pSourcePoser, HandPoser pDestinationPoser)

  Symmetrizes pSourcePoser and pDestinationPoser by copying all relevant data from pSourcePoser to pDestination↩ Poser.
- void SymmetrizeBendStateSwappers (BendStateSwapper pSourceSwapper, BendStateSwapper p↩ DestinationSwapper)

  Symmetrizes pSourceSwapper and pDestinationSwapper by copying all relevant data from pSourceSwapper to p↩ DestinationSwapper.

Public Attributes

- FlipAxes **flipBendAxes** = new FlipAxes() { x = false, y = false, z = false }

  What bend info axes should be flipped on the destination hands fingers?
- FlipAxes **flipBendAngleAxes** = new FlipAxes() { x = false, y = false, z = false }

  What bend info angle axes should be flipped on the dstination hands fingers?
- HandPoser **sourceHandPoser**

  The source HandPoser to copy from.
- HandPoser **destinationHandPoser**

  The destination KinematicHand to copy to.
- BendStateSwapper **sourceBendSwapper**

  The source BendStateSwapper to copy from.
- BendStateSwapper **destinationBendSwapper**

  The destination KinematicHand to copy to.

Events

- static Action< PoseSymmetryToolWindow > **Initialized**

  A C# delegate event that is invoked when the PoseSymmetryToolWindow is intialized.

5.32.1 Detailed Description

A window where adaptive hand components and settings from HnadPoser or BendStateSwapper components can be copied from a symmetrical hand.

Author: Mathew Aloisio

5.32.2 Member Function Documentation

5.32.2.1 SymmetrizeBendStateSwappers()

```
void AdaptiveHands.Editor.PoseSymmetryToolWindow.SymmetrizeBendStateSwappers (
            BendStateSwapper pSourceSwapper,
            BendStateSwapper pDestinationSwapper )
```

Symmetrizes pSourceSwapper and pDestinationSwapper by copying all relevant data from pSourceSwapper to pDestinationSwapper.

Parameters

pSourceSwapper	
pDestinationSwapper	

5.32.2.2 SymmetrizeHandPosers()

```
void AdaptiveHands.Editor.PoseSymmetryToolWindow.SymmetrizeHandPosers (
            HandPoser pSourcePoser,
            HandPoser pDestinationPoser )
```

Symmetrizes pSourcePoser and pDestinationPoser by copying all relevant data from pSourcePoser to p↩ DestinationPoser.

Parameters

pSourcePoser	
pDestinationPoser	

The documentation for this class was generated from the following file:

- PoseSymmetryToolWindow.cs

5.33 AdaptiveHands.Editor.EditorSymmetryUtility.ReplacementEntry Class Reference

Public Member Functions

- **ReplacementEntry** (string pLeftText, string pRightText, ReplaceMode pReplaceMode)

Public Attributes

- string **leftText**

  The left side symmetry identifier.
- string **rightText**

  The right side symmetry identifier.
- ReplaceMode **replaceMode**

  The ReplaceMode to use when replacing text.

The documentation for this class was generated from the following file:

- EditorSymmetryUtility.cs

5.34 AdaptiveHands.Editor.AdaptiveHandsEditorSettings.SettingsData Class Reference

Defines the settings for the AdaptiveHandsEditorSettings static class.

Public Attributes

- HandDiagramSettings **handDiagramSettings**

  The settings to use when displaying the hand diagram.
- HandleSettings **handleSettings**

  The settings to use for handles relating to adaptive hands.
- ExportAnimationSettings **exportAnimationSettings**

  The settings to use when exporting animations.

5.34.1 Detailed Description

Defines the settings for the AdaptiveHandsEditorSettings static class.

The documentation for this class was generated from the following file:

- AdaptiveHandsEditorSettings.cs

Index