Social Importance Asset

Generated by Doxygen 1.8.11

Fri May 20 2016 12:38:26

Contents

1 Namespace Index

1.1 Packages

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

2 Hierarchical Index

2.1 Class Hierarchy

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

3 Class Index

3.1 Class List

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

4 Namespace Documentation

4.1 ActionLibrary Namespace Reference

Namespaces

Classes

- interface IAction

  Interface used to represent an action execution request.

4.2 ActionLibrary.DTOs Namespace Reference

Classes

- class ActionDefinitionDTO

  Data Type Object Class for defining an Action.

4.3 KnowledgeBase Namespace Reference

Namespaces

4.4 KnowledgeBase.Conditions Namespace Reference

Enumerations

- enum LogicalQuantifier : byte { LogicalQuantifier.Existential, LogicalQuantifier.Universal }

  Represents logical quantification modes

4.4.1 Enumeration Type Documentation

4.4.1.1 enum KnowledgeBase.Conditions.LogicalQuantifier : byte [strong]

Represents logical quantification modes

Enumerator

Existential Sets of conditions evaluated in this mode, return true if at least on possible case is considered valid.

Universal Sets of conditions evaluated in this mode, return true only if all the possible cases are considered valid.

4.5 KnowledgeBase.DTOs Namespace Reference

Namespaces

4.6 KnowledgeBase.DTOs.Conditions Namespace Reference

Classes

- class ConditionSetDTO

  Data Type Object Class for the representation of a condition set

4.7 KnowledgeBase.WellFormedNames Namespace Reference

Classes

- class Name

  Well Formed Name Class.

4.8 SocialImportance Namespace Reference

Namespaces

Classes

- class SocialImportanceAsset

  Main class of the Social Importance Asset.

4.9 SocialImportance.DTOs Namespace Reference

Classes

- class AttributionRuleDTO

  Data Type Object Class for defining a Social Importance's Attribution Rule.
- class ClaimDTO

  Data Type Object Class for defining a Social Importance's Claim.
- class ConferralDTO

  Data Type Object Class for defining a Social Importance's Conferral action.
- class SocialImportanceDTO

  Data Type Object Class for defining a Social Importance Asset components.

5 Class Documentation

5.1 ActionLibrary.DTOs.ActionDefinitionDTO Class Reference

Data Type Object Class for defining an Action.

Inherited by SocialImportance.DTOs.ConferralDTO.

Properties

- Guid Id `[get, set]`
  The unique identifier of the action that this DTO is describing
- string Action `[get, set]`
  The action template, as a well formed string.
- string Target `[get, set]`
  The target of the action, if any.
- ConditionSetDTO Conditions `[get, set]`
  The set of conditions that must be true for this action execution.

5.1.1 Detailed Description

Data Type Object Class for defining an Action.

5.1.2 Property Documentation

5.1.2.1 string ActionLibrary.DTOs.ActionDefinitionDTO.Action `[get],[set]`

The action template, as a well formed string.

Attack([type],[strength])

5.1.2.2 ConditionSetDTO ActionLibrary.DTOs.ActionDefinitionDTO.Conditions `[get],[set]`

The set of conditions that must be true for this action execution.

5.1.2.3 Guid ActionLibrary.DTOs.ActionDefinitionDTO.Id `[get],[set]`

The unique identifier of the action that this DTO is describing

5.1.2.4 string ActionLibrary.DTOs.ActionDefinitionDTO.Target `[get],[set]`

The target of the action, if any.

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

- ActionDefinitionDTO.cs

5.2 ActionLibrary.IAction Interface Reference

Interface used to represent an action execution request.

Properties

- Name ActionName [get]

  The name of the action to execute
- Name Target [get]

  The target of the action, if appliable
- IList< Name > Parameters [get]

  The parameters values that the action needs in order for it to be executed. The parameter order is equal to the order of the variables in the action template, defined in the correspondent ActionDefinitionDTO.

5.2.1 Detailed Description

Interface used to represent an action execution request.

5.2.2 Property Documentation

5.2.2.1 Name ActionLibrary.IAction.ActionName [get]

The name of the action to execute

5.2.2.2 IList<Name> ActionLibrary.IAction.Parameters [get]

The parameters values that the action needs in order for it to be executed. The parameter order is equal to the order of the variables in the action template, defined in the correspondent ActionDefinitionDTO.

5.2.2.3 Name ActionLibrary.IAction.Target [get]

The target of the action, if appliable

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

- IAction.cs

5.3 KnowledgeBase.DTOs.Conditions.ConditionSetDTO Class Reference

Data Type Object Class for the representation of a condition set

Public Attributes

- string[ ] ConditionSet

  The conditions to be evaluated as a single set.

Properties

- LogicalQuantifier Quantifier `[get, set]`

  The logical quantifier of this condition set. Used to change how the entier condition set is evaluated.

5.3.1 Detailed Description

Data Type Object Class for the representation of a condition set

5.3.2 Member Data Documentation

5.3.2.1 string [ ] KnowledgeBase.DTOs.Conditions.ConditionSetDTO.ConditionSet

The conditions to be evaluated as a single set.

5.3.3 Property Documentation

5.3.3.1 LogicalQuantifier KnowledgeBase.DTOs.Conditions.ConditionSetDTO.Quantifier `[get],[set]`

The logical quantifier of this condition set. Used to change how the entier condition set is evaluated.

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

- ConditionSetDTO.cs

5.4 KnowledgeBase.WellFormedNames.Name Class Reference

Well Formed Name Class.

Inherits IGroundable< Name >, IComparable< Name >, IPerspective< Name >, and ICloneable.

Public Member Functions

- abstract Name GetFirstTerm ()

  Returns the first term of this Name. Primitive and Variable Names will always return them selfs.
- abstract IEnumerable< Name > GetTerms ()

  Return all terms contained inside this Name.
- abstract Name GetNTerm (int index)

  Return the term at the specified index.
- abstract IEnumerable< Name > GetLiterals ()

  Generates a sequence of all Names contained inside this Name.
- abstract IEnumerable< Name > GetVariables ()

  Generates a sequence of all variables contained inside this Name.
- abstract bool HasGhostVariable ()

  Tells if this name contains a Ghost variable
- abstract bool HasSelf ()

Tells if this name contains a SELF primitive.

- bool ContainsVariable (Name variable)

  Verifies if a specific variable is contained inside this Name.
- Name ApplyPerspective (Name name)

  Swaps every instance of the given Name with the SELF primitive.
- Name RemovePerspective (Name name)

  Swaps every instance of the SELF primitive with the given Name.
- abstract Name SwapPerspective (Name original, Name newName)

  Swaps every instance of the requested Name with another.
- abstract Name MakeGround (SubstitutionSet bindings)

  Given a SubstitutionSet, tries to ground this Name by substituting every variable with the corresponding value.
- abstract Name ReplaceUnboundVariables (string id)

  Adds a tag to the end of every variable inside this Name, effectively modifying their identifier.
- abstract Name RemoveBoundedVariables (string id)

  Removes a tag from the end of every variable inside this Name, effectively modifying their identifier.
- abstract object Clone ()

  Clones this Name, returning an equal copy. If this clone is changed afterwards, the original object remains the same.
- abstract bool Match (Name name)

  Determines if this matches the given name template. Both Names are matched to each other if all their Symbols are equal to one another or if a Symbol matches a universal Symbol.
- abstract Name ApplyToTerms (Func< Name, Name > transformFunction)

  Apply a transformation function to this Name. The function will receive every term of this name, and should return a name to be swapped with the old one.

Static Public Member Functions

- static Name GenerateUniqueGhostVariable ()

  Creates a new Name, representing a variable without a proper human readable identifier. Usefull to create temporary substitution variables.
- static operator Name (string definition)

  Explicit cast from a string to a Name. Similar from calling Name.Build(string)
- static bool operator== (Name n1, Name n2)

  Name comparison operator. Tells if two names are equal to one another.
- static bool operator!= (Name n1, Name n2)

  Name comparison operator. Tells if two names are diferent from one another.
- static Name BuildName (Name rootTerm, Name firstTerm, params Name[ ] otherTerms)

  Creates a composed Name, using two or more Names
- static Name BuildName (IEnumerable< Name > terms)

  Creates a Name, using a sequence of Names.
- static Name BuildName (string str)

  Creates a new Name instance by parsing a string.

Public Attributes

- const string NIL_STRING = "-"

  *The string representation of a **NIL** value Name.*
- const string SELF_STRING = "SELF"

  *The string representation of the **"SELF"** primitive Name.*
- const string UNIVERSAL_STRING = "*"

  The string representation of the Universal matching Name.

- readonly bool IsUniversal

  Tells if this is name the Universal Matching Symbol
- readonly bool IsConstant

  Tells if this name does not contain universal or variable Symbols
- readonly bool IsVariable

  Tells if this name is a variable definition
- readonly bool IsPrimitive

  Tells if this name is a primitive value
- readonly bool IsComposed

  Tells if this name is a composition of other names

Static Public Attributes

- static readonly Name NIL_SYMBOL

  A constant containing an instance of a NIL Name
- static readonly Name SELF_SYMBOL

  A constant containing an instance of a SELF Name
- static readonly Name UNIVERSAL_SYMBOL

  A constant containing an instance of a Universal matching Name

Properties

- bool IsGrounded `[get]`

  Tells if this name is grounded. A grounded Name is one that do not contain variables.
- abstract int NumberOfTerms `[get]`

  The number of terms that compose this name. Primitive and Variable Names will always return 1.

5.4.1 Detailed Description

Well Formed Name Class.

A well formed name is used to specify goal/action names, objects, properties, constants, and relations.

Its syntax is based on first order logic symbols, variables and predicates.

Names can be generated from strings, or from composition with other names. All names are case-insensitive.

Even though the Name is class type, its underlying behaviour is similar to a value type structure. This means that every modification to its values, returns a new instance of a Name object, preserving the state of the original one.

By default, Names separated in the following categories:

- Primitives
  - John
  - Dog
  - Blue
  - 34.5
- Variables
  - [x]
  - [strength]
- Composed Names
  - Color(Sky)
  - Likes(John)
  - Size(Ball)
  - Kick(Hard, Low)

5.4.2 Member Function Documentation

5.4.2.1 Name KnowledgeBase.WellFormedNames.Name.ApplyPerspective (Name *name*)

Swaps every instance of the given Name with the SELF primitive.

Parameters

name	The Name instance to swap from.

Returns

A new instance, which is a clone of this Name, but with every instance of the given Name swaped with SELF.

5.4.2.2 abstract Name KnowledgeBase.WellFormedNames.Name.ApplyToTerms (Func< Name, Name > *transformFunction*) [pure virtual]

Apply a transformation function to this Name. The function will receive every term of this name, and should return a name to be swapped with the old one.

Parameters

transformFunction	The function we want to apply to this Name.

Returns

A new Name instance, which is the original one with the transformed function applied.

5.4.2.3 static Name KnowledgeBase.WellFormedNames.Name.BuildName (Name *rootTerm,* Name *firstTerm,* params Name[ ] *otherTerms*) [static]

Creates a composed Name, using two or more Names

Parameters

rootTerm	The Name that will be root of the composed Name.
firstTerm	The first term of the composed Name.
otherTerms	The remaining terms of the composed Name.

Exceptions

ArgumentException	Thrown if the rootTerm is not a primitive Name.

5.4.2.4 static Name KnowledgeBase.WellFormedNames.Name.BuildName (IEnumerable< Name > *terms*) [static]

Creates a Name, using a sequence of Names.

Parameters

terms	The Name set used to generate the new one.

Exceptions

ArgumentException	Thrown if the first element of the set is not a primitive Name.

5.4.2.5 static Name KnowledgeBase.WellFormedNames.Name.BuildName (string *str*) [static]

Creates a new Name instance by parsing a string.

Parameters

str	The string to parse.

Exceptions

ArgumentException	Thrown if the given string is empty.

5.4.2.6 abstract object KnowledgeBase.WellFormedNames.Name.Clone ( ) [pure virtual]

Clones this Name, returning an equal copy. If this clone is changed afterwards, the original object remains the same.

Returns

The Name's copy.

5.4.2.7 bool KnowledgeBase.WellFormedNames.Name.ContainsVariable (Name *variable*)

Verifies if a specific variable is contained inside this Name.

Parameters

variable	The variable Name we want to verify

Exceptions

ArgumentException	Thrown if the given argument is not a variable definition.

5.4.2.8 static Name KnowledgeBase.WellFormedNames.Name.GenerateUniqueGhostVariable ( ) [static]

Creates a new Name, representing a variable without a proper human readable identifier. Usefull to create temporary substitution variables.

5.4.2.9 abstract Name KnowledgeBase.WellFormedNames.Name.GetFirstTerm ( ) [pure virtual]

Returns the first term of this Name. Primitive and Variable Names will always return them selfs.

5.4.2.10 abstract IEnumerable<Name> KnowledgeBase.WellFormedNames.Name.GetLiterals ( ) [pure virtual]

Generates a sequence of all Names contained inside this Name.

5.4.2.11 abstract Name KnowledgeBase.WellFormedNames.Name.GetNTerm (int *index*) [pure virtual]

Return the term at the specified index.

Parameters

index	The zero-based index of the term to get.

Exceptions

IndexOutOfRangeException	Thrown if the given index is out of bounds.

- For Primitive or Variable Names, any index different from 0, will throw an IndexOutOfRangeException.
- Using this method with a 0 index is the same as using GetFirstTerm()

5.4.2.12 abstract IEnumerable<Name> KnowledgeBase.WellFormedNames.Name.GetTerms ( ) [pure virtual]

Return all terms contained inside this Name.

5.4.2.13 abstract IEnumerable<Name> KnowledgeBase.WellFormedNames.Name.GetVariables ( ) [pure virtual]

Generates a sequence of all variables contained inside this Name.

5.4.2.14 abstract bool KnowledgeBase.WellFormedNames.Name.HasGhostVariable ( ) [pure virtual]

Tells if this name contains a Ghost variable

GenerateUniqueGhostVariable()

5.4.2.15 abstract bool KnowledgeBase.WellFormedNames.Name.HasSelf ( ) [pure virtual]

Tells if this name contains a SELF primitive.

5.4.2.16 abstract Name KnowledgeBase.WellFormedNames.Name.MakeGround (SubstitutionSet *bindings*) [pure virtual]

Given a SubstitutionSet, tries to ground this Name by substituting every variable with the corresponding value.

Parameters

bindings	The SubstitutionSet to be used to ground this Name.

Returns

A new instance, which is a clone of this Name, but grounded as much as possible.

- If this instance is already grounded before calling this method, it will just return the same Name.
- This method does not warrant that this Name will be fully grounded, as the given SubstitutionSet may not contain the substitution variables needed to perform the task.

5.4.2.17 abstract bool KnowledgeBase.WellFormedNames.Name.Match (Name *name*) [pure virtual]

Determines if this matches the given name template. Both Names are matched to each other if all their Symbols are equal to one another or if a Symbol matches a universal Symbol.

Parameters

name	The Name to match with this instance.

Returns

True if both Names match with each other, false otherwise.

5.4.2.18 static KnowledgeBase.WellFormedNames.Name.operator Name (string *definition*) [explicit], [static]

Explicit cast from a string to a Name. Similar from calling Name.Build(string)

5.4.2.19 static bool KnowledgeBase.WellFormedNames.Name.operator!= (Name *n1*, Name *n2*) [static]

Name comparison operator. Tells if two names are diferent from one another.

5.4.2.20 static bool KnowledgeBase.WellFormedNames.Name.operator== (Name *n1*, Name *n2*) [static]

Name comparison operator. Tells if two names are equal to one another.

5.4.2.21 abstract Name KnowledgeBase.WellFormedNames.Name.RemoveBoundedVariables (string *id*) [pure virtual]

Removes a tag from the end of every variable inside this Name, effectively modifying their identifier.

Parameters

id	The tag to remove from every variable.

Returns

A new instance, which is a clone of this Name, but with every variable identifier changed in order to exclude the requested tag.

///

- If this instance is already grounded before calling this method, it will just return the same Name.
- The tag is only removed if, and only if, the variable identifier ends with the requested tag.

5.4.2.22 Name KnowledgeBase.WellFormedNames.Name.RemovePerspective (Name *name*)

Swaps every instance of the SELF primitive with the given Name.

Parameters

name	The Name instance to swap to.

Returns

A new instance, which is a clone of this Name, but with every instance of SELF swaped with the given Name.

5.4.2.23 abstract Name KnowledgeBase.WellFormedNames.Name.ReplaceUnboundVariables (string *id*) [pure virtual]

Adds a tag to the end of every variable inside this Name, effectively modifying their identifier.

Parameters

id	The tag to add to every variable.

Returns

A new instance, which is a clone of this Name, but with every variable identifier changed in order to include the new tag.

///

- If this instance is already grounded before calling this method, it will just return the same Name.

5.4.2.24 abstract Name KnowledgeBase.WellFormedNames.Name.SwapPerspective (Name *original*, Name *newName*) [pure virtual]

Swaps every instance of the requested Name with another.

Parameters

original	The Name instance to swap from.
newName	The Name instance to swap to.

Returns

A new instance, which is a clone of this Name, but with every instance of the original Name swaped with the new one.

5.4.3 Member Data Documentation

5.4.3.1 readonly bool KnowledgeBase.WellFormedNames.Name.IsComposed

Tells if this name is a composition of other names

5.4.3.2 readonly bool KnowledgeBase.WellFormedNames.Name.IsConstant

Tells if this name does not contain universal or variable Symbols

5.4.3.3 readonly bool KnowledgeBase.WellFormedNames.Name.IsPrimitive

Tells if this name is a primitive value

5.4.3.4 readonly bool KnowledgeBase.WellFormedNames.Name.IsUniversal

Tells if this is name the Universal Matching Symbol

5.4.3.5 readonly bool KnowledgeBase.WellFormedNames.Name.IsVariable

Tells if this name is a variable definition

5.4.3.6 const string KnowledgeBase.WellFormedNames.Name.NIL_STRING = "-"

The string representation of a **NIL** value Name.

5.4.3.7 const string KnowledgeBase.WellFormedNames.Name.SELF_STRING = "SELF"

The string representation of the **"SELF"** primitive Name.

5.4.3.8 const string KnowledgeBase.WellFormedNames.Name.UNIVERSAL_STRING = "*"

The string representation of the Universal matching Name.

5.4.4 Property Documentation

5.4.4.1 bool KnowledgeBase.WellFormedNames.Name.IsGrounded [get]

Tells if this name is grounded. A grounded Name is one that do not contain variables.

5.4.4.2 abstract int KnowledgeBase.WellFormedNames.Name.NumberOfTerms [get]

The number of terms that compose this name. Primitive and Variable Names will always return 1.

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

- Name.cs

5.5 SocialImportance.DTOs.AttributionRuleDTO Class Reference

Data Type Object Class for defining a Social Importance's Attribution Rule.

Public Attributes

- string Target

  The condition variable that represents the target name in the rule condition set.
- int Value

  The value to be attributed to the target, if all conditions are valid.
- ConditionSetDTO Conditions

  The condition set used to validate this rule.

5.5.1 Detailed Description

Data Type Object Class for defining a Social Importance's Attribution Rule.

Attribution rules are used to define conditions that when validated through the asset's beliefs will attribute to the target a Social Importance Value. The total Social Importance Value of a target is given by the sum of all valid Attribution rules in the asset's definition.

5.5.2 Member Data Documentation

5.5.2.1 ConditionSetDTO SocialImportance.DTOs.AttributionRuleDTO.Conditions

The condition set used to validate this rule.

5.5.2.2 string SocialImportance.DTOs.AttributionRuleDTO.Target

The condition variable that represents the target name in the rule condition set.

5.5.2.3 int SocialImportance.DTOs.AttributionRuleDTO.Value

The value to be attributed to the target, if all conditions are valid.

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

- AttributionRuleDTO.cs

5.6 SocialImportance.DTOs.ClaimDTO Class Reference

Data Type Object Class for defining a Social Importance's Claim.

Public Attributes

- string ActionTemplate

  The action's name template used for action matching.
- uint ClaimSI

  The maximum Social Importance value the action's target can have, before the action is considered socially unacceptable.

5.6.1 Detailed Description

Data Type Object Class for defining a Social Importance's Claim.

Claims are used to tell if actions are socialy acceptable. Socialy aceptable actions are ones that the Claim value don't exced the action's target Social Importance value.

This can be used to filter agent's possible actions remaining only the socially accepted ones, or determine if action targeting the agent is socialy accepted or not, acording to the agent's beliefs.

5.6.2 Member Data Documentation

5.6.2.1 string SocialImportance.DTOs.ClaimDTO.ActionTemplate

The action's name template used for action matching.

5.6.2.2 uint SocialImportance.DTOs.ClaimDTO.ClaimSI

The maximum Social Importance value the action's target can have, before the action is considered socially unacceptable.

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

- ClaimDTO.cs

5.7 SocialImportance.DTOs.ConferralDTO Class Reference

Data Type Object Class for defining a Social Importance's Conferral action.

Inherits ActionLibrary.DTOs.ActionDefinitionDTO.

Properties

- uint ConferralSI `[get, set]`

  The Conferral's social importance value.

5.7.1 Detailed Description

Data Type Object Class for defining a Social Importance's Conferral action.

Conferral actions are ones that an agent might want to execute, but only if the action's target's Social Importance value is bellow or equal to the Conferral's Social Importance.

If multiple conferrals are available for execution, the asset will only select the one with the highest social importance value.

Conferrals are bonded by Claims, like any other action, and as such even if a conferral can be executed, if its target's social importance value it's above an action Claim, it will not execute.

See also

ClaimDTO

5.7.2 Property Documentation

5.7.2.1 uint SocialImportance.DTOs.ConferralDTO.ConferralSI `[get],[set]`

The Conferral's social importance value.

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

- ConferralDTO.cs

5.8 SocialImportance.DTOs.SocialImportanceDTO Class Reference

Data Type Object Class for defining a Social Importance Asset components.

Public Attributes

- AttributionRuleDTO[ ] AttributionRules
  The set of attribution rules used to calculate Social importance values to targets
- ClaimDTO[ ] Claims
  The set of Claims used to determine if a action is socially acceptable.
- ConferralDTO[ ] Conferral
  The set of Conferrals we want the asset to executed.

5.8.1 Detailed Description

Data Type Object Class for defining a Social Importance Asset components.

5.8.2 Member Data Documentation

5.8.2.1 AttributionRuleDTO [ ] SocialImportance.DTOs.SocialImportanceDTO.AttributionRules

The set of attribution rules used to calculate Social importance values to targets

5.8.2.2 ClaimDTO [ ] SocialImportance.DTOs.SocialImportanceDTO.Claims

The set of Claims used to determine if a action is socially acceptable.

5.8.2.3 ConferralDTO [ ] SocialImportance.DTOs.SocialImportanceDTO.Conferral

The set of Conferrals we want the asset to executed.

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

- SocialImportanceDTO.cs

5.9 SocialImportance.SocialImportanceAsset Class Reference

Main class of the Social Importance Asset.

Inherits LoadableAsset< SocialImportanceAsset >, and ICustomSerialization.

Public Member Functions

- void BindEmotionalAppraisalAsset (EmotionalAppraisalAsset eaa)

  Binds an Emotional Appraisal Asset (EAA) to this Social Importance Asset instance. Without a EAA instance binded to this asset, social importance evaluations will not work. InvalidateCachedSI() is automatically called by this method.
- float GetSocialImportance (string target, string perspective="self")

  Calculate the Social Importance value of a given target, in a particular perspective. If no perspective is given, the current agent's perspective is used as default.
- void InvalidateCachedSI ()

  Clears all cached Social Importance values, allowing new values to be recalculated uppon request.
- IAction DecideConferral (string perspective)

  Request a conferral action from the Social Importance Asset.
- IEnumerable< IAction > FilterActions (string perspective, IEnumerable< IAction > actionsToFilter)

  Filters a set of actions using the defined Social Importance Claims.
- void LoadFromDTO (SocialImportanceDTO dto)

  Load a Social Importance Asset definition from a DTO object.
- SocialImportanceDTO GetDTO ()

  Returns a DTO containing all the asset's configurations.

Properties

- EmotionalAppraisalAsset LinkedEA `[get]`

  The Emotional Appraisal Asset that is binded to this Social Importance Asset instance

5.9.1 Detailed Description

Main class of the Social Importance Asset.

New dynamic properties available by this asset uppon binding it with a Emotional Appraisal Asset:

- SI([target])
  - Gives the Social Importance value attributed to the given target

5.9.2 Member Function Documentation

5.9.2.1 void SocialImportance.SocialImportanceAsset.BindEmotionalAppraisalAsset (EmotionalAppraisalAsset *eaa*)

Binds an Emotional Appraisal Asset (EAA) to this Social Importance Asset instance. Without a EAA instance binded to this asset, social importance evaluations will not work. InvalidateCachedSI() is automatically called by this method.

Parameters

eaa	The Emotional Appraisal Asset to be binded to this asset.

5.9.2.2 IAction SocialImportance.SocialImportanceAsset.DecideConferral (string *perspective*)

Request a conferral action from the Social Importance Asset.

The action will be generated based on the defined Conferrals, and will always be the maximum valued conferral that can still respect the asset's defined Claims.

Parameters

perspective	From which perspective do we want to generate the action. If the perspective is diferent from "self", it will be like the asset predicting which action will be executed by the entity defined in the perspective.

Returns

The action we want to execute or predict.

5.9.2.3 IEnumerable<IAction> SocialImportance.SocialImportanceAsset.FilterActions (string *perspective,* IEnumerable< IAction > *actionsToFilter*)

Filters a set of actions using the defined Social Importance Claims.

Parameters

perspective	From which perspective do we want to filter the actions. If the perspective is diferent from "self", it will be like the asset is evaluating desirable action from another entity's point of view.
actionsToFilter	The set of actions we want to filter.

Returns

The set of filtered actions. No action returned will have a Claim value higher that the Social Importance attributed to the target of the action.

ClaimDTO

5.9.2.4 SocialImportanceDTO SocialImportance.SocialImportanceAsset.GetDTO ( )

Returns a DTO containing all the asset's configurations.

5.9.2.5 float SocialImportance.SocialImportanceAsset.GetSocialImportance (string *target,* string *perspective =* `"self"`)

Calculate the Social Importance value of a given target, in a particular perspective. If no perspective is given, the current agent's perspective is used as default.

All values calculated by this method are automatically cached, in order to optimize future searches. If the values are needed to be recalculated, call InvalidateCachedSI() to clear the cached values.

Parameters

target	The name of target which we want to calculate the SI
perspective	From which perspective do we want to calculate de SI.

Returns

The value of Social Importance attributed to given target by the perspective of a particular agent.

5.9.2.6 void SocialImportance.SocialImportanceAsset.InvalidateCachedSI ( )

Clears all cached Social Importance values, allowing new values to be recalculated uppon request.

5.9.2.7 void SocialImportance.SocialImportanceAsset.LoadFromDTO (SocialImportanceDTO *dto*)

Load a Social Importance Asset definition from a DTO object.

Use this to procedurally configure the asset.

Parameters

dto	The DTO containing the data to load

5.9.3 Property Documentation

5.9.3.1 EmotionalAppraisalAsset SocialImportance.SocialImportanceAsset.LinkedEA [get]

The Emotional Appraisal Asset that is binded to this Social Importance Asset instance

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

- SocialImportanceAsset.cs

Index