# MindEngine

Generated by Doxygen 1.8.11

# Contents

# Chapter 1

# Todo List

**Member Database::IdType**

Use identifier with strings may decrease performance with large systems due to string-matching operations. Will change this to integer or unsigned integer as soon as I find a way to match identifier with human-readable strings.

**Member Rules::DataGroupMatch::matchesNode (const Database::DataNode ∗node, void ∗bindings)**

Add check for wildcard identifier with strings.

Add check for wildcards here.

**Member Rules::Rule::action ()=0**

Examine the performance and reusability of using a method. If it is hard to expand, find a way to use a struct/class instead.

# Chapter 2

# Namespace Index

## 2.1 Namespace List

Here is a list of all documented namespaces with brief descriptions:

# Chapter 3

# Hierarchical Index

## 3.1 Class Hierarchy

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

# Chapter 4

# Class Index

## 4.1 Class List

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

# Chapter 5

# Namespace Documentation

## 5.1 Database Namespace Reference

Contains classes to represent Rule-based system's database, which stores knowledge available to the AI agent.

### Classes

- class DataGroup

  *Represents a non-leaf node, which contains children. Its children can be any DataNode object: either another DataGroup, or a Datum only.*
- class DataNode

  *Base class of each node in the database tree. Since every node needs an identifier, but non-leaf nodes contain their children, while leaf nodes store values.*
- class Datum

  *A Datum consists of an identifier and and value. In Database's tree structure, a leaf node is a Datum.*

### Typedefs

- typedef std::string IdType

  *Currently set the type of the ID of data nodes as string.*

### 5.1.1 Detailed Description

Contains classes to represent Rule-based system's database, which stores knowledge available to the AI agent.

### 5.1.2 Typedef Documentation

#### 5.1.2.1 typedef std::string **Database::IdType**

Currently set the type of the ID of data nodes as string.

**Todo** Use identifier with strings may decrease performance with large systems due to string-matching operations. Will change this to integer or unsigned integer as soon as I find a way to match identifier with human-readable strings.

## 5.2 Rules Namespace Reference

Contains the implementation of rules as well as the mechanism to match the rules and the data in the database.

### Classes

- struct DataGroupMatch

  *Matches a group of data in the database. This is done by building a match data structure that mirrors the data structure that is being searched for in the database.*
- struct DataNodeMatch

  *A struct derived from Match, it is responsible for matching a single DataNode in the database.*
- struct Match

  *Provides the mechanism to match the data item from the rule with any item inside the database.*
- class Rule

  *Represent a rule in a Rule-based system. A rule has two components: an if clause is going to be used to match against the database and a function to perform any action required.*

### 5.2.1 Detailed Description

Contains the implementation of rules as well as the mechanism to match the rules and the data in the database.

# Chapter 6

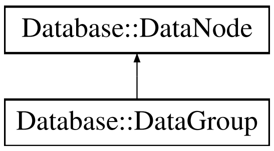
# Class Documentation

## 6.1 Database::DataGroup Class Reference

Represents a non-leaf node, which contains children. Its children can be any DataNode object: either another DataGroup, or a Datum only.

```
#include <DataGroup.h>
```

Inheritance diagram for Database::DataGroup:



### Public Member Functions

- DataGroup ()

  *DataGroup default constructor.*
- DataGroup (const std::string &identifier, DataNode ∗rightSibling, DataNode ∗leftMostChild)

  *DataGroup constructor.*
- virtual ∼DataGroup ()

  *∼DataGroup destructor.*
- DataNode ∗ getLeftMostChild () const

  *Data nodes are put into a left-most child, right sibling tree. This function returns the pointer to the left most child of this node.*
- bool isGroup ()

  *Allows user to check whether this node is a DataGroup or not.*

### Additional Inherited Members

### 6.1.1 Detailed Description

Represents a non-leaf node, which contains children. Its children can be any DataNode object: either another DataGroup, or a Datum only.

**See also**

DataNode
Datum

### 6.1.2 Constructor & Destructor Documentation

#### 6.1.2.1 Database::DataGroup::DataGroup ( const std::string & *identifier*, DataNode ∗ *rightSibling*, DataNode ∗ *leftMostChild* )

DataGroup constructor.

**Parameters**

| | |
|---|---|
| *identifier* | a string parameter. |
| *rightSibling* | a DataNode pointer representing this node's right sibling in tree. |
| *leftMostChild* | a DataNode pointer representing this node's left most child in tree. |

### 6.1.3 Member Function Documentation

#### 6.1.3.1 DataNode ∗ Database::DataGroup::getLeftMostChild ( ) const

Data nodes are put into a left-most child, right sibling tree. This function returns the pointer to the left most child of this node.

**Returns**

The left most child of this data group node.

#### 6.1.3.2 bool Database::DataGroup::isGroup ( )

Allows user to check whether this node is a DataGroup or not.

**Returns**

true if this node is a DataGroup, otherwise returns false.

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

- Engine/Database/DataGroup.h
- Engine/Database/DataGroup.cpp

## 6.2 Rules::DataGroupMatch Struct Reference

Matches a group of data in the database. This is done by building a match data structure that mirrors the data structure that is being searched for in the database.

```
#include <DataGroupMatch.h>
```

## Public Member Functions

- virtual bool matchesNode (const Database::DataNode ∗node, void ∗bindings)

  *Tries to match the given data node from the database against the criteria. Method used: a recursive matching algorithm that travels down the given node, trying to match it against the structure of this match and its children.*

## Public Attributes

- Database::IdType identifier

  *The identifier to match.*
- DataNodeMatch ∗ leftMostChild

  *The first sub-match in this group.*

### 6.2.1 Detailed Description

Matches a group of data in the database. This is done by building a match data structure that mirrors the data structure that is being searched for in the database.

### 6.2.2 Member Function Documentation

#### 6.2.2.1 bool Rules::DataGroupMatch::matchesNode ( const Database::DataNode ∗ *node*, void ∗ *bindings* ) `[virtual]`

Tries to match the given data node from the database against the criteria. Method used: a recursive matching algorithm that travels down the given node, trying to match it against the structure of this match and its children.

**Parameters**

| | |
|---|---|
| *node* | The database to match on. |
| *bindings* | When part of the if clause matches a wild card, it is added to the bindings. This parameter is both input and output parameter. |

**Returns**

true if matches, otherwise return false.

**Todo** Add check for wildcard identifier with strings.

**Todo** Add check for wildcards here.

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

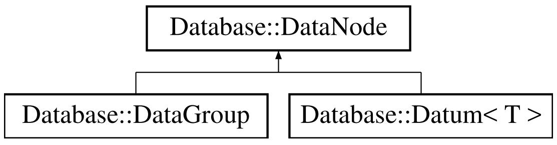
- Engine/Rules/DataGroupMatch.h
- Engine/Rules/DataGroupMatch.cpp

## 6.3 Database::DataNode Class Reference

Base class of each node in the database tree. Since every node needs an identifier, but non-leaf nodes contain their children, while leaf nodes store values.

```
#include <DataNode.h>
```

Inheritance diagram for Database::DataNode:



### Public Member Functions

- DataNode ()

  *DataNode default constructor.*
- DataNode (const IdType &identifier, DataNode ∗rightSibling)

  *DataNode constructor with parameters.*
- virtual ∼DataNode ()

  *∼DataNode destructor.*
- IdType getIdentifier () const

  *The data nodes have unique identifiers.*
- DataNode ∗ getRightSibling () const

  *Data nodes are put into a left-most child, right sibling tree. This function is used to get the sibling next to current node.*
- virtual bool isGroup () const

  *Allows user to check whether this node is a DataGroup or not.*
- virtual bool isDatum () const

  *Allows user to check whether this node is a Datum or not.*

### Protected Attributes

- IdType identifier

  *Each item of data should be identified to know what the value means.*
- DataNode ∗ rightSibling

  *The right sibling node of this node, or NULL if this node is the right most child.*

### 6.3.1 Detailed Description

Base class of each node in the database tree. Since every node needs an identifier, but non-leaf nodes contain their children, while leaf nodes store values.

**See also**

DataGroup
Datum

### 6.3.2 Constructor & Destructor Documentation

#### 6.3.2.1 Database::DataNode::DataNode ( const IdType & *identifier,* DataNode ∗ *rightSibling* )

DataNode constructor with parameters.

**Parameters**

| | |
|---|---|
| *identifier* | an ID parameter. |
| *rightSibling* | a DataNode pointer representing this node's right sibling in tree. structure |

### 6.3.3 Member Function Documentation

#### 6.3.3.1 IdType Database::DataNode::getIdentifier ( ) const

The data nodes have unique identifiers.

**Returns**

The identifier of this node.

#### 6.3.3.2 DataNode $*$ Database::DataNode::getRightSibling ( ) const

Data nodes are put into a left-most child, right sibling tree. This function is used to get the sibling next to current node.

**Returns**

The right sibling node of this node, or NULL if this node is the right most child.

#### 6.3.3.3 bool Database::DataNode::isDatum ( ) const [virtual]

Allows user to check whether this node is a Datum or not.

**Returns**

true if this node is a Datum, otherwise returns false.

#### 6.3.3.4 bool Database::DataNode::isGroup ( ) const [virtual]

Allows user to check whether this node is a DataGroup or not.

**Returns**

true if this node is a DataGroup, otherwise returns false.

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

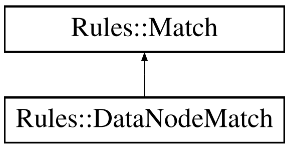
- Engine/Database/DataNode.h
- Engine/Database/DataNode.cpp

## 6.4 Rules::DataNodeMatch Struct Reference

A struct derived from Match, it is responsible for matching a single DataNode in the database.

`#include <DataNodeMatch.h>`

Inheritance diagram for Rules::DataNodeMatch:



### Public Member Functions

- virtual bool matches (const Database::DataNode ∗database, void ∗bindings)

  *Matches the given database by checking each element in the database against the matchesNode method.*
- bool matchesChildren (const Database::DataGroup ∗group, void ∗bindings)

  *Matches all the children of the given group to see if any of them pass the matchesNode test. This is used in the implementation of the matches method to iterate through items in the database.*
- virtual bool matchesNode (const Database::DataNode ∗node, void ∗bindings)=0

  *Matches the data node from the database against the criteria in this match.*

### Public Attributes

- DataNodeMatch ∗ rightSibling

  *The right sibling of this node in match tree.*

### 6.4.1 Detailed Description

A struct derived from Match, it is responsible for matching a single DataNode in the database.

Conceptually, the match class could match the whole database in a single operation: it is only ever fed the whole database, so it can do with that what it likes. However in practical, the vast majority of matching requirements involve trying to find a single item in the database. This struct's match method iterates through the items in the database, and calls matchesNode on each one. Matches items can be implemented in sub-classes to check if the item fulfils the mtach criteria.

Data node matches are arranged into tree, just like how data nodes are.

**See also**

Match

### 6.4.2 Member Function Documentation

#### 6.4.2.1 bool Rules::DataNodeMatch::matches ( const Database::DataNode ∗ *database,* void ∗ *bindings* ) `[virtual]`

Matches the given database by checking each element in the database against the matchesNode method.

**Parameters**

| | |
|---|---|
| *database* | The database to match on. |
| *bindings* | When part of the if clause matches a wild card, it is added to the bindings. This parameter is both input and output parameter. |

**Returns**

true if matches, else returns false.

Implements Rules::Match.

#### 6.4.2.2 bool Rules::DataNodeMatch::matchesChildren ( const Database::DataGroup ∗ *group*, void ∗ *bindings* )

Matches all the children of the given group to see if any of them pass the matchesNode test. This is used in the implementation of the matches method to iterate through items in the database.

**Parameters**

| | |
|---|---|
| *group* | The group to match on. |
| *bindings* | When part of the if clause matches a wild card, it is added to the bindings. This parameter is both input and output parameter. |

**Returns**

true if matches, else return false.

#### 6.4.2.3 virtual bool Rules::DataNodeMatch::matchesNode ( const Database::DataNode ∗ *node*, void ∗ *bindings* ) `[pure virtual]`

Matches the data node from the database against the criteria in this match.

**Parameters**

| | |
|---|---|
| *node* | The database to match on. |
| *bindings* | When part of the if clause matches a wild card, it is added to the bindings. This parameter is both input and output parameter. |

**Returns**

true if matches, else return false.

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

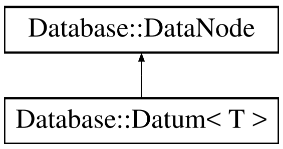
- Engine/Rules/DataNodeMatch.h
- Engine/Rules/DataNodeMatch.cpp

## 6.5 Database::Datum< T > Class Template Reference

A Datum consists of an identifier and and value. In Database's tree structure, a leaf node is a Datum.

```
#include <Datum.h>
```

Inheritance diagram for Database::Datum< T >:



### Public Member Functions

- Datum (T value)

    *Datum constructor.*
- Datum (const std::string &identifier, DataNode ∗rightSibling, T value)

    *Datum constructor.*
- virtual ∼Datum ()

    *∼Datum destructor*
- void setValue (T newValue)

    *To change the value of the Datum.*
- T getValue () const

    *To get the value of the Datum.*
- bool isDatum ()

    *Allows user to check whether this node is a Datum or not.*

### Additional Inherited Members

### 6.5.1 Detailed Description

**template<class T>**
**class Database::Datum< T >**

A Datum consists of an identifier and and value. In Database's tree structure, a leaf node is a Datum.

**See also**

DataNode
DataGroup

### 6.5.2 Constructor & Destructor Documentation

#### 6.5.2.1 template<class T > Database::Datum< T >::Datum ( T *value* )

Datum constructor.

**Parameters**

| | |
|---|---|
| *value* | the value that the Datum holds. |

#### 6.5.2.2 template<class T > Database::Datum< T >::Datum ( const std::string & *identifier*, DataNode ∗ *rightSibling*, T *value* )

Datum constructor.

**Parameters**

| | |
|---|---|
| *identifier* | a string parameter. |
| *rightSibling* | a DataNode pointer representing this node's right sibling in tree. |
| *value* | the value that the Datum holds. |

### 6.5.3 Member Function Documentation

#### 6.5.3.1 template<class T > T Database::Datum< T >::getValue ( ) const

To get the value of the Datum.

**Returns**

The value that the datum is currently holding.

#### 6.5.3.2 template<class T > bool Database::Datum< T >::isDatum ( )

Allows user to check whether this node is a Datum or not.

**Returns**

true if this node is a Datum, otherwise returns false.

#### 6.5.3.3 template<class T > void Database::Datum< T >::setValue ( T *newValue* )

To change the value of the Datum.

**Parameters**

| | |
|---|---|
| *newValue* | The new value that is going to be assigned to the Datum. |

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

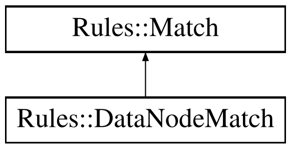
- Engine/Database/Datum.h
- Engine/Database/Datum.cpp

## 6.6 Rules::Match Struct Reference

Provides the mechanism to match the data item from the rule with any item inside the database.

```
#include <Match.h>
```

Inheritance diagram for Rules::Match:



### Public Member Functions

- virtual bool matches (const Database::DataNode ∗database, void ∗bindings)=0
  *Check the match on the database.*

### 6.6.1 Detailed Description

Provides the mechanism to match the data item from the rule with any item inside the database.

### 6.6.2 Member Function Documentation

#### 6.6.2.1 virtual bool Rules::Match::matches ( const Database::DataNode ∗ *database,* void ∗ *bindings* ) `[pure virtual]`

Check the match on the database.

**Parameters**

| | |
|---|---|
| *database* | The database to match on. |
| *bindings* | When part of the if clause matches a wild card, it is added to the bindings. This parameter is both input and output parameter. |

**Returns**

true if matches, else returns false.

Implemented in Rules::DataNodeMatch.

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

- Engine/Rules/Match.h

## 6.7 Rules::Rule Class Reference

Represent a rule in a Rule-based system. A rule has two components: an if clause is going to be used to match against the database and a function to perform any action required.

```
#include <Rule.h>
```

### Public Member Functions

- virtual void action ()=0

  *The action is going to be carried out when the rule matches.*

### Public Attributes

- Match ∗ ifClause

  *Consist of a set of data items, in a similar format to those in the database.*

### 6.7.1 Detailed Description

Represent a rule in a Rule-based system. A rule has two components: an if clause is going to be used to match against the database and a function to perform any action required.

### 6.7.2 Member Function Documentation

#### 6.7.2.1 virtual void Rules::Rule::action ( ) `[pure virtual]`

The action is going to be carried out when the rule matches.

**Todo** Examine the performance and reusability of using a method. If it is hard to expand, find a way to use a struct/class instead.

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

- Engine/Rules/Rule.h

# Index