
Roboy Dialog Manager

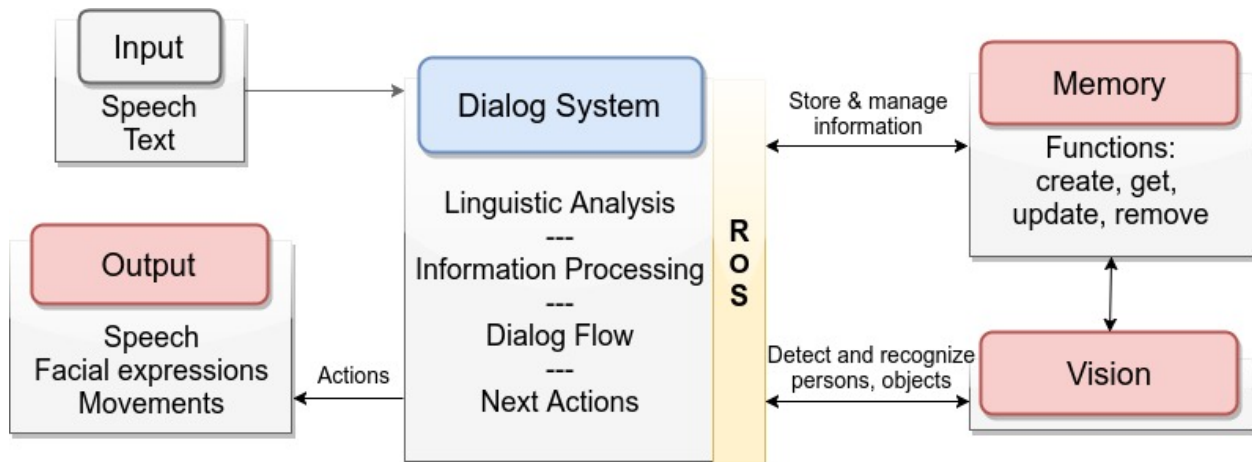
Release 0.0.

Jan 25, 2018

Usage and Installation

1	Status	3
2	Contents:	5

This project aims to implement human-like conversation routines for the humanoid anthropomorphic robot Roboy. The overview diagram shows the external systems which Dialog System interacts with, and the tasks for which the system is responsible.



CHAPTER 1

Status

Stable functionality:

- Roboy introduces himself
- Roboy answers questions about himself
- Roboy recognizes once someone says his name
- Roboy asks questions people he meets

In development:

- Roboy stores and recalls information (name, occupation, ect.) about people he meets
- Roboy recognizes the intent behind an asked questions (age, creator, capabilities etc.)

CHAPTER 2

Contents:

2.1 Installation

We use Apache Maven build automation tool.

2.1.1 Using command line

Install Maven on the computer. `sudo apt install maven`

Clone the Dialog Manager repository. `git clone https://github.com/Roboy/DialogSystem`

Navigate to the root module. `cd DialogSystem`

Compile the project - Maven will take care of the rest. `mvn compile`

Execute the project. `mvn exec:java -Dexec.mainClass="robey.dialog.DialogSystem"`

2.1.2 Using IDE (Eclipse, IntelliJ IDEA)

Clone the Dialog Manager repository. `git clone https://github.com/Roboy/DialogSystem`

Now, import Dialog System as a Maven project into the IDE of your choice. Build and execute using `robey.dialog.DialogSystem` as the main class.

2.2 Getting started

2.2.1 How does it work?

The basic NLP architecture is designed as a pipeline.

1. An input device (derived from `de.robey.io.InputDevice`) is producing text.

2. The text is passed to a variety of linguistic analyzers (derived from `de.robey.linguistics.sentenceanalysis.Analyzer`). This currently consists of a Tokenizer and a POS tagger (both in `de.robey.linguistics.sentenceanalysis.SentenceAnalyzer`) but could in the future be accompanied by named entity recognition, a syntactical and semantical analysis, an interpretation of the sentence type or other tools.
3. The results of all these linguistics analyzers are collected together with the original text and stored in an Interpretation instance. (`de.robey.linguistics.sentenceanalysis.Interpretation`)
4. The interpretation is passed on to a state machine (states are derived from `de.robey.dialog.personality.states.State`) within a personality class (derived from `de.robey.dialog.personality.Personality`) that decides how to react to the utterance. In the future, the intentions (`de.robey.logic.Intention`) determined by the state machine will then be formulated into proper sentences or other actions (`de.robey.dialog.action.Action`) by a module called Verbalizer. Currently, these actions are still directly created in the personality class.
5. Finally, the created actions are sent to the corresponding output device (`de.robey.io.OutputDevice`).

2.2.2 Design choices

There are interfaces for each step in the processing pipeline to enable an easy exchange of elements. The goal would be to easily exchange personalities based on the occasion.

The implementation of the pipeline is in Java. Integrations with tools in other languages, like C++ RealSense stuff, should be wrapped in a module in the pipeline.

2.2.3 How to extend it?

Pick the corresponding interface, depending on which part of the system you want to extend. If you want to add new devices go for the input or output device interfaces. If you want to extend the linguistic analysis implement the Analyzer interface or extend the SentenceAnalyzer class. If you are happy with input, linguistics and output and just want to create more dialog, implement the Personality interface.

Create a new ...	By implementing ...
Input Device	<code>de.robey.io.InputDevice</code>
NLP Analyzer	<code>de.robey.linguistics.sentenceanalysis.Analyzer</code>
State Machine	<code>de.robey.dialog.personality.Personality</code>
State	<code>de.robey.dialog.personality.states.State</code>
Action type	<code>de.robey.dialog.action.Action</code>
Output device	<code>de.robey.io.OutputDevice</code>

The interfaces are deliberately simple, containing only 0 - 2 methods that have to be implemented. Once you implemented your new classes include them in the personality used in `de.robey.dialog.DialogSystem`, if you only implemented single states or directly in `de.robey.dialog.DialogSystem` for everything else.

2.3 Personality and states

2.3.1 Overview

To enable a natural way of communication, Roboy's Dialog Module implements a flexible architecture using personality classes, which each manage a number of different states. This enables us to dynamically react to clues from the

conversation partner and spontaneously switch between purposes and stages of a dialog, mimicing a natural conversation.

2.3.2 Personality

During one run-through, the Dialog System uses a single Personality instance (`de.robey.dialog.personality.Personality`). A personality is designed to define how robey reacts to every given situation, and as such Robey can always only represent one personality at a time. Different personalities are meant to be used in different situations, like a more formal or loose one depending on the occasion.

The current primary personality is the `SmallTalkPersonality` (`de.robey.dialog.personality.SmallTalkPersonality`).

A new personality (`robey.newDialog.StateBasedPersonality`) is currently being implemented. Similarly to the `SmallTalkPersonality`, it is built on top of a state machine. The new personality is designed to be more generic one and allows to load the behaviour from a personality file. The file stores a representation of the state machine. Additionally, it is still possible to define the dialog structure directly from code (as it was done in previous implementations).

2.3.3 Legacy State

A state's activity can be divided into two stages. When the state is entered, the initial action from the `act()` method is carried out, which is expected to trigger a response from the person. After Roboy has received and analyzed the response, the `react()` method completes the current state's actions and Roboy moves on to the next state.

The `AbstractBooleanState` describes a special case where the state's reaction depends on whether the `act()` method resulted in successful interaction. States which implement `AbstractBooleanState` can respond differently move on into different stages according to their `determineSuccess()` method.

For example, the initial action of `de.robey.dialog.personality.states.IntroductionState` is to ask the user's name. Then the response is analyzed externally and when the state's `determineSuccess()` method is called, it checks whether a name was extracted. If this is the case, then the system outputs predefined sentences with the extracted name embedded into them. Otherwise, fallback sentences are used which do not include a name.

2.3.4 New State

Currently, we are improving the state system. Old state implementations will be replaced with newer ones. The functionality of the `AbstractBooleanState` will be improved to allow arbitrary conditional transitions in every new state. Nested states will be replaced with the fallback concept.

A state's activity can be divided into two stages. When the state is entered, the initial action from the `act()` method is carried out, which is expected to trigger a response from the person. After Roboy has received and analyzed the response, the `react()` method completes the current state's actions and Roboy picks a transition to the next state defined by the `getNextState()` method of the current state.

It is possible to remain in the same state for many cycles. In this case the `getNextState()` method just returns a reference to the current state (`this`) and the `act()` and `react()` methods are carried out again.

A state can have any number of transitions to other states. Every transition has a name (like "next" or "errorState"). When changing states, the following state can be selected based on internal conditions of the current state. For example, a state can expect a "yes/no" answer and have tree outgoing transitions: "gotYes", "gotNo" and "askAgain" (if the reply is not "yes/no").

When designing a new state, only the transition names are defined. The following states are defined in the personality file later. At run time the state machine loads the file and initializes the transitions to point it correct states. The destination state can be retrieved by the transition name using `getTransition(transitionName)`.

A fallback can be attached to a state. In the case this state doesn't know how to react to an utterance, it can return `null` from the `react()` function. The state machine will query the fallback in this case. This concept helps to decouple the states and reduce the dependencies between them. When implementing the `react()` function of a new state, it is sufficient to detect unknown input and return `null`.

2.3.5 Legacy State machine structure

Every state defines at least one successor state, and more complex hierarchies can be realized - for example as a fallback system for cases when a single state cannot respond in a meaningful manner. The fallback system implemented using nested states in the legacy state machine and will be improved in the newer implementation. The following is an example from the documentation of `SmallTalkPersonality`:

The current legacy state machine looks like this:

Greeting state

V

Introduction state

V

Question Randomizer state

└_Question Answering state

└_Segue state └_Wild talk state

The Question Randomizer, Question Answering, Segue and Wilk talk states are nested. If one cannot give an appropriate reaction to the given utterance, the utterance is passed on to the next one. The Wild talk state will always answer.

2.4 The memory module

2.4.1 General design

To remember information about itself and its conversation partners, their hobbies, occupations and origin, a persistent Memory module has been implemented using the Neo4j graph database.

2.4.2 Implementation

Roboy's Dialog System interactions with the Memory module are based on ROS messages. The messages are sent using the methods in `de.robey.ros.RosMainNode`, which implements the four query types based on the specified Memory services:

Method name	Description
CreateMemoryQuery	Creates a node in Memory database
UpdateMemoryQuery	Adds or changes information of an existing node
GetMemoryQuery	Retrieves either one node or an array of IDs
DeleteMemoryQuery	Removes information from or deletes a node
CypherMemoryQuery	For more complex queries (future)

The messages received from Memory are in JSON format. To enable flexible high-level handling of Memory information, two classes were created to incorporate the node structures and logic inside the Dialog System. The `de.roboy.memory.nodes.MemoryNodeModel` contains the labels, properties and relationships in a format which can be directly parsed from and into JSON. For this, Dialog is using the GSON parsing methods which enable direct translation of a JSON String into its respective Java class representation.

Methods such as `getRelation()` or `setProperties()` were implemented to allow intuitive handling of the `MemoryNodeModel` instances. A separate class, `de.roboy.memory.nodes.Interlocutor`, encapsulates a `MemoryNodeModel` and is intended to further ease saving information about the current conversation partner of Roboy. `Interlocutor` goes one step further by also abstracting the actual calls to memory, such that adding the name of the conversant performs an automatic lookup in the memory with subsequent updating of the person-related information. This is then available in all subsequent interactions, such that Roboy can refrain from asking questions twice, or refer to information he remembers from earlier conversations.

2.5 Configuration

The Dialog Manager can be called with a specific system configuration that determines which external services will be used within the session. The `ROS_HOSTNAME` is set through the `config.properties` file at project root.

2.5.1 Usage

Set profile in the execution invocation like this: `mvn exec:java -Dexec.mainClass="roboy.dialog.DialogSystem" -Dprofile=NOROS`

If running from within an IDE, edit the run configurations to include the profile as VM option:
`-Dprofile=NOROS`

Without a specified profile, `DEFAULT` will be used. Please note that this profile requires setting a valid `ROS_HOSTNAME` address in the `config.properties` file to function properly! If ROS is not set up, use the `NOROS` profile to prevent the Dialog Manager from using ROS-dependent services.

2.5.2 Profiles

Profile	Description
DEFAULT	Used when no other profile is set, assumes that all requirements (ROS, Internet connection, speakers, mic) are fulfilled.
NOROS	To be used when ROS services are not set up, avoids calls to memory, speech synthesis, voice output, etc.
STAN-DALONE	To be used when running without Internet connection - this profile includes all restrictions of NOROS and also does not call DBPedia.
MEMORY-ONLY	To be used during Memory development, when no other ROS services are running. Only Neo4j-related ROS calls will be made.
DEBUG	With this setting, DM will run like DEFAULT but not shut down when ROS failures are encountered.

2.5.3 Extending

To extend or change the configurations, have a look at the instructions in the `roboy.dialog.Config` class.

2.6 Context

The goal of Context is to collect information about Roboy's environment and state. This information can be used by the DM classes and also to react upon situations that match certain conditions, such as turning the head of Roboy when the Interlocutor moves.

2.6.1 Architecture

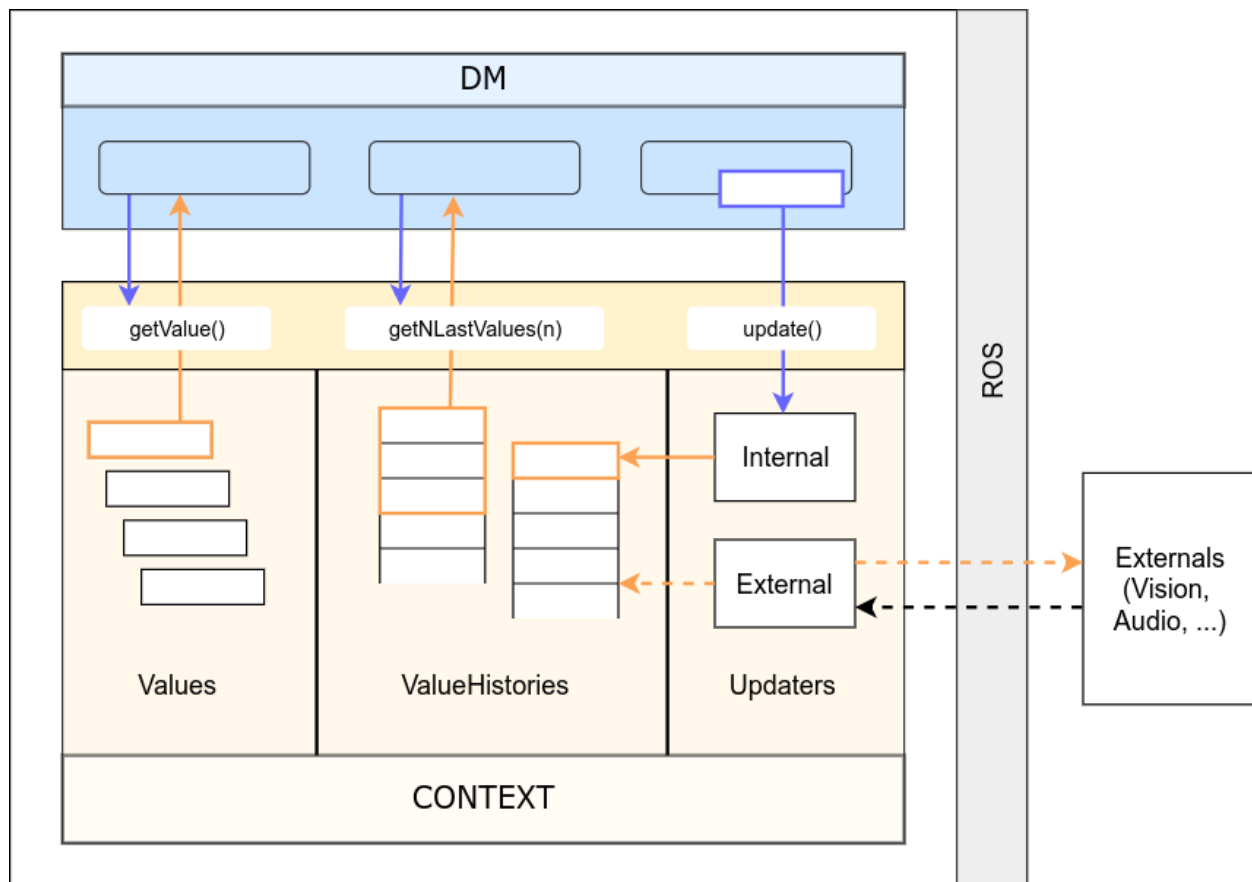
alt Context architecture

The Context supports storing data as a `Value` or `ValueHistory`. A `Value` only stores the latest data object that was pushed to it. A `ValueHistory` stores every value it receives and assigns each a unique key, thus the values can be ordered by their adding time.

2.6.2 How to add Values?

Here we describe how a new `Value` can be created and added to the Context. Sample implementations can be found inside `roboy.context.contextObjects` package.

1. Consider what type of data will be stored in the `Value`. For this example, we chose `String`.
2. In the `contextObjects` directory, create a new class which inherits from the `Value` class. The final signature should look similar to: `public class SampleValue extends Value<String>` (replacing `String` with your type).
3. **Add the new `Value` in the main class `Context.java`:**
 - (a) In the standard constructor, initialize the `Value` object and add it to the builder of the value map:
`put(SampleValue.class, sampleValue)`
 - (b) Make the value available over the enum `Values` within the `Context` class by adding a new element with your class name and stored data type. For example: `SAMPLE_VALUE(SampleValue.class, String.class);`



2.6.3 How to add ValueHistories?

ValueHistories extend the functionality of Values by storing all data objects sent to them. Over the `getNLastValues(int n)` method, a map with several most recent data objects can be retrieved, including their ordering.

Adding a ValueHistory is very much alike to adding a Value, just make sure to:

1. extend `ValueHistory<>` instead of `Value<>`,
2. in `Context.java`, add the new object to the Builder of valueHistories instead of values, and to the enum `ValueHistories` instead of `Values`.

2.6.4 How to add Updaters?

New values can only be added to the Context over an Updater instance. Internal updaters can be used by DM classes to actively add new values. External updaters run in separate threads and seek out new values, for example over a ROS connections to the Vision module.

Adding an External Updater

Currently, the only implementation of an external updater is the `IntervalUpdater` abstract class. Usage:

1. Create a class extending `IntervalUpdater` and implement its `update()` method. It should retrieve the values and finally add them over the `target.updateValue(value)` method call.
2. Add the updater to `externalUpdaters` in the `Context.java` constructor, setting its `target` parameter with the `Value` or `ValueHistory` object created in the same constructor.

Adding a new Internal Updater

1. Create a class extending `InternalUpdater<targetClass, valueType>`. The class and data type of the target `Value` or `ValueHistory` are the generic parameters for the updater.
2. A constructor is required for the class, simply match the `InternalUpdater` constructor and call `super(target)` within. An example is in the `DialogTopicsUpdater` class.
3. In the `Context` class constructor, initialize the updater and add it to the `internalUpdaters` map.
4. Add an entry to the `Updaters` enum, similarly as discussed above for `Values`.

2.7 Semantic Parser

Semantic parser is used to translate text representation into formal language representation. The aim is to be able to process user utterances and react upon them.

`robey_parser` is based on *SEMPRE* <<http://nlp.stanford.edu/software/sempre/>>. It is currently being modified to fulfill Roboy Dialog system needs.

2.7.1 Installation

In order to use semantic parser, you need to:

- clone `robey_parser` repository:


```
git clone http://github.com/Roboy/roboy_parser
```

- navigate to created repository:

```
cd roboy_parser
```

- build it:

```
ant freebase
```

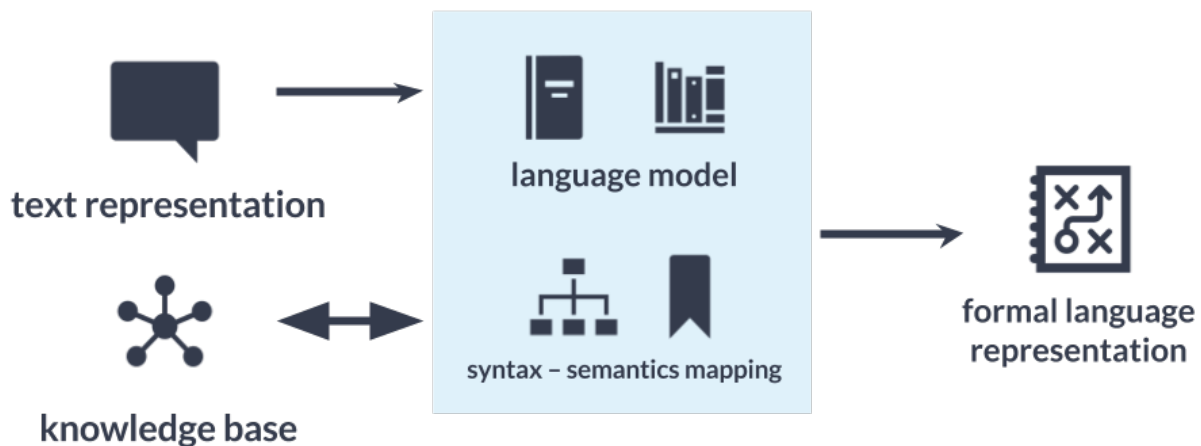
- run it:

```
./quick_start [options]
```

2.7.2 Architecture

Semantic parser is based on the language model and NLP algorithms that then apply rules to the utterance to translate it. Language model consists of: - set of grammar rules, - lexicon, - training dataset.

General architecture can be seen on the diagram below.



alt Semantic parser general architecture

2.7.3 Implementation

robey_parser is a separate Java project and is communicating using `WebSocket`. Dialog system has a client implemented in `SemanticParserAnalyzer.java` class. It is therefore part of Natural Language Understanding unit.

Functionalities

Roboy parser currently has currently following functionalities:

Table 2.1: Semantic Parser algorithms used

Functionality	Software used	Summary
POS Tags	OpenNLP	Tagging tokens as part of speech
NER Tags	OpenNLP	Tool used to tag named entities like PERSON, NUMBER, ORGANIZATION
Triple extraction	OpenIE	Tool used to extract triples from sentences in form (Subject, Predicate, Object)

2.7.4 Usage

In order to run the parser, you need to run **roboy_parser** first - see instructions on *project Github* <http://github.com/Roboy/roboy_parser and then run Dialog System.

Configurations

To test parser, you can run following configurations using `quick_start.sh` script. For more information refer to *project documentation* <http://github.com/Roboy/roboy_parser

Table 2.2: Possible parser configurations

Command	Options
<code>./quick_start</code>	Default configuration. Using custom Roboy grammar and lexicons
<code>./quick_start freebase</code>	Example setup to test Freebase functionality

2.8 API

class AbstractBooleanState

Abstract super class for states that fork between two possible subsequent states.

The `determineSuccess` method needs to be implemented by subclass to determine if the success or failure state should be moved into next.

Inherits from *roboy.dialog.personality.states.State*

Subclassed by *roboy.dialog.personality.states.ConverseState*, *roboy.dialog.personality.states.GenerativeCommunicationState*, *roboy.dialog.personality.states.GreetingState*, *roboy.dialog.personality.states.IdleState*, *roboy.dialog.personality.states.InquiryState*, *roboy.dialog.personality.states.IntroductionState*, *roboy.dialog.personality.states.LocationDBpedia*, *roboy.dialog.personality.states.PersonalFollowUpState*, *roboy.dialog.personality.states.PersonalQAState*

Public Functions

State `roboy.dialog.personality.states.AbstractBooleanState.getSuccess()`

void `roboy.dialog.personality.states.AbstractBooleanState.setSuccess(State success)`

Sets the state Roboy moves into if the `determineSuccess` method returns true.

Parameters

- `success`: The following state

```
State roboy.dialog.personality.states.AbstractBooleanState.getFailure()
```

```
void roboy.dialog.personality.states.AbstractBooleanState.setFailure(State failure)
```

Sets the state Roboy moves into if the determineSuccess method returns false.

Parameters

- *failure*: The following state

```
void roboy.dialog.personality.states.AbstractBooleanState.setNextState(State state)
```

```
void roboy.dialog.personality.states.AbstractBooleanState.setSuccessTexts(List< String
```

```
void roboy.dialog.personality.states.AbstractBooleanState.setFailureTexts(List< String
```

```
Reaction roboy.dialog.personality.states.AbstractBooleanState.react(Interpretation inp
```

Protected Functions

```
abstract boolean roboy.dialog.personality.states.AbstractBooleanState.determineSuccess
```

Needs to be implemented by subclasses.

If the method returns true the state machine moves to the success state, if it returns false it moves to the failure state.

Return true or false depending on the examined condition of the method

Parameters

- *input*: The interpretation of all inputs

Protected Attributes

```
State roboy.dialog.personality.states.AbstractBooleanState.success
```

```
State roboy.dialog.personality.states.AbstractBooleanState.failure
```

Private Members

```
List<String> roboy.dialog.personality.states.AbstractBooleanState.successTexts = Lists.strin
```

```
List<String> roboy.dialog.personality.states.AbstractBooleanState.failureTexts = Lists.strin
```

```
template <V>
```

```
interface AbstractValue
```

Stores a single value.

On update, the value is overwritten.

Subclassed by *roboy.context.AbstractValueHistory< K, V >*, *roboy.context.Value< V >*

Public Functions

```
V roboy.context.AbstractValue< V >.getValue()
```

```
void roboy.context.AbstractValue< V >.updateValue(V key)
```

```
template <K, V>
```

interface AbstractValueHistory

ValueHistory maintains a map containing all (current and past) values.

These values are accessible over the `getLastNValues` method.

Inherits from *roboy::context::AbstractValue< V >*

Public Functions

Map<K, V> roboy.context.AbstractValueHistory< K, V >.getLastNValues (int n)

interface Action

The marker interface for an action.

The interface is empty, since different output devices will require different informations in an action. The most important action is the *SpeechAction* which is used for communication.

Subclassed by *roboy.dialog.action.FaceAction*, *roboy.dialog.action.ShutDownAction*, *roboy.dialog.action.SpeechAction*

interface Analyzer

All linguistic analyses implement the *Analyzer* interface.

An analyzer always takes an existing interpretation of a sentence and returns one including its own analysis results (usually an enriched version of the input interpretation).

Subclassed by *roboy.linguistics.sentenceanalysis.AnswerAnalyzer*, *roboy.linguistics.sentenceanalysis.DictionaryBasedSentenceTy*, *roboy.linguistics.sentenceanalysis.EmotionAnalyzer*, *roboy.linguistics.sentenceanalysis.IntentAnalyzer*, *roboy.linguistics.sentenceanalysis.OntologyNERAnalyzer*, *roboy.linguistics.sentenceanalysis.OpenNLPParser*, *roboy.linguistics.sentenceanalysis.OpenNLPPPOSTagger*, *roboy.linguistics.sentenceanalysis.Preprocessor*, *roboy.linguistics.sentenceanalysis.SemanticParserAnalyzer*, *roboy.linguistics.sentenceanalysis.SentenceAnalyzer*, *roboy.linguistics.sentenceanalysis.SimpleTokenizer*

Public Functions

Interpretation roboy.linguistics.sentenceanalysis.Analyzer.analyze (Interpretation sent

class AnecdoteState

Utters the given text and moves to the given state.

Used for telling anecdotes.

Inherits from *roboy.dialog.personality.states.State*

Public Functions

roboy.dialog.personality.states.AnecdoteState.AnecdoteState (State nextState, String an

List<Interpretation> roboy.dialog.personality.states.AnecdoteState.act ()

Reaction roboy.dialog.personality.states.AnecdoteState.react (Interpretation input)

Private Members

State roboy.dialog.personality.states.AnecdoteState.nextState

String roboy.dialog.personality.states.AnecdoteState.anecdote

class AnswerAnalyzer

Checks the predicate argument structures produced by the *OpenNLPParser* analyzer and looks for possible answers to questions in them.

It creates the outputs *Linguistics.OBJ_ANSWER* for situations where the answer to the question is in the object of the sentence (e.g. “Frank” in the sentence “I am Frank” to the question “Who are you?”) and *Linguistics.PRED_ANSWER* if it is in the predicate or in the predicate and the object combined (e.g. “swimming” in the answer “I like swimming” to the question “What is your hobby?”).

Inherits from *roboy.linguistics.sentenceanalysis.Analyzer*

Public Functions

Interpretation `roboy.linguistics.sentenceanalysis.AnswerAnalyzer.analyze(Interpretation)`

class AnswerAnalyzerTest**Public Functions**

```
void roboy.linguistics.sentenceanalysis.AnswerAnalyzerTest.testName()
void roboy.linguistics.sentenceanalysis.AnswerAnalyzerTest.testOccupation()
void roboy.linguistics.sentenceanalysis.AnswerAnalyzerTest.testOrigin()
void roboy.linguistics.sentenceanalysis.AnswerAnalyzerTest.testHobby()
void roboy.linguistics.sentenceanalysis.AnswerAnalyzerTest.testMovie()
```

Private Functions

```
String roboy.linguistics.sentenceanalysis.AnswerAnalyzerTest.analyze(String sentence)
String roboy.linguistics.sentenceanalysis.AnswerAnalyzerTest.analyzePred(String sentence)
```

Private Static Attributes

```
final SimpleTokenizer roboy.linguistics.sentenceanalysis.AnswerAnalyzerTest.tokenizer =
final OpenNLPPPOSTagger roboy.linguistics.sentenceanalysis.AnswerAnalyzerTest.pos = new O
final OpenNLPPParser roboy.linguistics.sentenceanalysis.AnswerAnalyzerTest.parser = new O
final AnswerAnalyzer roboy.linguistics.sentenceanalysis.AnswerAnalyzerTest.answer = new A
```

template <H, V>

class AttributeManager

The collection of values, split into valueHistories (H) and single values (V).

Protected Attributes

```
ImmutableClassToInstanceMap<AbstractValueHistory> roboy.context.AttributeManager< H ex
ImmutableClassToInstanceMap<AbstractValue> roboy.context.AttributeManager< H extends E
```

Package Functions

`protected<T> T roboy.context.AttributeManager< H extends ExternalContextInterface, V e`

`protected<K, T> Map<K, T> roboy.context.AttributeManager< H extends ExternalContextInt`

`protected<T> T roboy.context.AttributeManager< H extends ExternalContextInterface, V e`

class BingInput

Using Bing to perform speech to text.

Requires internet connection.

Inherits from *roboy.io.InputDevice*

Public Functions

`roboy.io.BingInput.BingInput (RosMainNode node)`

`Input roboy.io.BingInput.listen()`

Private Members

`RosMainNode roboy.io.BingInput.rosMainNode`

class BingOutput

Uses Bing for text to speech.

Requires internet connection.

Inherits from *roboy.io.OutputDevice*

Public Functions

`void roboy.io.BingOutput.act (List< Action > actions)`

`void roboy.io.BingOutput.say (String text)`

class CelebritySimilarityInput

Should perform the celebrity look-a-like vision input.

Isn't implemented yet.

Inherits from *roboy.io.InputDevice*

Public Functions

`Input roboy.io.CelebritySimilarityInput.listen()`

class CelebrityState

Checks whether it was asked 'Whom do I look like' or something similar and answers with the most similar celebrity if it was detected by vision.

If no trigger sentence was used, the given inner state is executed instead.

Inherits from *roboy.dialog.personality.states.State*

Public Functions

```

roboy.dialog.personality.states.CelebrityState.CelebrityState(State inner)
void roboy.dialog.personality.states.CelebrityState.setTop(State top)
List<Interpretation> roboy.dialog.personality.states.CelebrityState.act()
Reaction roboy.dialog.personality.states.CelebrityState.react(Interpretation input)

```

Private Members

```

State roboy.dialog.personality.states.CelebrityState.inner
State roboy.dialog.personality.states.CelebrityState.top

```

Private Static Attributes

```

final List<String> roboy.dialog.personality.states.CelebrityState.triggerSentences = List
final List<String> roboy.dialog.personality.states.CelebrityState.formulations = Lists.strin

```

class CerevoiceOutput

Cerevoice text to speech.

Inherits from *roboy.io.OutputDevice*

Public Functions

```

roboy.io.CerevoiceOutput.CerevoiceOutput(RosMainNode node)
void roboy.io.CerevoiceOutput.act(List< Action > actions)
void roboy.io.CerevoiceOutput.say(String text)

```

Private Members

```

RosMainNode roboy.io.CerevoiceOutput.rosMainNode

```

class CommandLineCommunication

Inherits from *roboy.io.Communication*

Public Functions

```

void roboy.io.CommandLineCommunication.setPersonality(Personality p)
void roboy.io.CommandLineCommunication.communicate()

```

Private Functions

```

void roboy.io.CommandLineCommunication.talk(List< Action > actions)

```

Private Members

`Personality roboy.io.CommandLineCommunication.personality`

`SentenceAnalyzer roboy.io.CommandLineCommunication.analyzer`

class `CommandLineInput`

Uses the command line as input device.

Inherits from *roboy.io.InputDevice*

Public Functions

`Input roboy.io.CommandLineInput.listen()`

Protected Functions

`void roboy.io.CommandLineInput.finalize()`

Private Members

`Scanner roboy.io.CommandLineInput.sc = new Scanner(System.in)`

class `CommandLineOutput`

Uses the command line as output device.

Inherits from *roboy.io.OutputDevice*

Public Functions

`void roboy.io.CommandLineOutput.act(List< Action > actions)`

interface `Communication`

Subclassed by *roboy.io.CommandLineCommunication*

Public Functions

`void roboy.io.Communication.setPersonality(Personality p)`

`void roboy.io.Communication.communicate()`

class `Concept`

Protege memory concept.

Public Functions

`roboy.util.Concept.Concept()`

`roboy.util.Concept.Concept(Map< String, Object > attrs)`

`roboy.util.Concept.Concept(String name)`

`void roboy.util.Concept.addAttribute(String property, Object value)`


```

void roboy.util.Concept.addAttributes(Map< String, Object > attrs)
Map<String, Object> roboy.util.Concept.getAttributes()
Object roboy.util.Concept.getAttribute(String key)
String roboy.util.Concept.getProperties()
String roboy.util.Concept.getValues()
boolean roboy.util.Concept.hasAttribute(String property)
Object roboy.util.Concept.retrieve()
boolean roboy.util.Concept.updateInMemory()
int roboy.util.Concept.getID()

```

Private Members

```

Map<String, Object> roboy.util.Concept.attributes
class Concept

```

Package Attributes

```

String roboy.linguistics.Concept.id
class Config

```

Save runtime configurations (profiles) for Roboy.

1) Configuration variables define alternating behaviors. 2) To create a combination of configurations, add a new profile to Configurations and implement its setProfile method.

Public Functions

```

roboy.dialog.Config.Config(ConfigurationProfile profile)
    Constructor switching to the correct profile.

```

Public Static Functions

```

static ConfigurationProfile roboy.dialog.Config.getProfileFromEnvironment(String profileString)
    ConfigurationProfile instance which matches the profileString.

```

Parameters

- profileString: String value of the configuration profile name.

Public Static Attributes

```

boolean roboy.dialog.Config.STANDALONE = false
    If true, Roboy avoids using network-based services such as DBpedia as well as ROS.

```

```

boolean roboy.dialog.Config.NOROS = false
    If true, Roboy avoids using ROS-based services.

```

boolean roboy.dialog.Config.SHUTDOWN_ON_ROS_FAILURE = true

If true, Roboy will not continue executing if the ROS main node fails to initialize.

boolean roboy.dialog.Config.SHUTDOWN_ON_SERVICE_FAILURE = true

If true, Roboy will not continue executing if any of the ROS services failed to initialize.

String roboy.dialog.Config.ROS_HOSTNAME = null

ROS hostname, will be fetched from the configuration file in the DEFAULT profile.

boolean roboy.dialog.Config.MEMORY = true

If true, memory will be queried.

Ensure that if NOROS=false, then MEMORY=true. When NOROS=true, MEMORY can be either true or false.

int roboy.dialog.Config.PARSER_PORT = -1

Semantic parser socket port.

boolean roboy.dialog.Config.DEMO_GUI = false

Context GUI demo trigger.

Set manually, if wanted.

Private Functions

void roboy.dialog.Config.setDefaultProfile()

void roboy.dialog.Config.setNoROSProfile()

void roboy.dialog.Config.setStandaloneProfile()

void roboy.dialog.Config.setDebugProfile()

void roboy.dialog.Config.setMemoryProfile()

void roboy.dialog.Config.initializeYAMLConfig()

Private Members

YAMLConfiguration roboy.dialog.Config.yamlConfig

Private Static Attributes

String roboy.dialog.Config.yamlConfigFile = "config.properties"

Configuration file to store changing values.

enum ConfigurationProfile

List of profile names.

The variables are set in the corresponding set<name>Profile() method. String values make it possible to define the profile in start command with: -Dprofile=<profileString>

Public Functions

roboy.dialog.Config.ConfigurationProfile.ConfigurationProfile(String profile)

Public Members

```
roboy.dialog.Config.ConfigurationProfile.DEFAULT = ("DEFAULT")
roboy.dialog.Config.ConfigurationProfile.NOROS = ("NOROS")
roboy.dialog.Config.ConfigurationProfile.STANDALONE = ("STANDALONE")
roboy.dialog.Config.ConfigurationProfile.DEBUG = ("DEBUG")
roboy.dialog.Config.ConfigurationProfile.MEMORY = ("MEMORY")
String roboy.dialog.Config.ConfigurationProfile.profileName
```

class Context

Singleton class serving as an interface to access all context objects.

Takes care of correct initialization of attribute histories and updaters.

Queries to values are handled through the inherited *AttributeManager* methods.

For usage examples, check out ContextTest.java

Inherits from *roboy.context.AttributeManager< Context.ValueHistories, Context.Values >*

Public Members

```
final HashMap<Class, InternalUpdater> roboy.context.Context.internalUpdaters = new HashM
```

Public Static Functions

```
static Context roboy.context.Context.getInstance ()
```

Package Functions

```
public<V> void roboy.context.Context.updateValue (Updaters updater, V value)
```

Directly update an attribute.

Parameters

- updater: The name of the *Value* or *ValueHistory* object.
- value: Data to put into the *Value* or *ValueHistory* object.

Private Functions

```
roboy.context.Context.Context ()
```

Private Members

```
final ArrayList<ExternalUpdater> roboy.context.Context.externalUpdaters = new ArrayList<>()
```

Private Static Attributes

```
Context roboy.context.Context.context
```

class ContextGUI

A simple GUI with the goal of showing the attribute values and histories in the *Context*.

Public Static Functions

```
static void roboy.context.GUI.ContextGUI.run()
```

Private Functions

```
roboy.context.GUI.ContextGUI.ContextGUI()
void roboy.context.GUI.ContextGUI.prepareGUI()
void roboy.context.GUI.ContextGUI.startFrame()
void roboy.context.GUI.ContextGUI.updateValues()
void roboy.context.GUI.ContextGUI.updateHistories()
```

Private Members

```
JFrame roboy.context.GUI.ContextGUI.mainFrame
TitledBorder roboy.context.GUI.ContextGUI.valueBorder
JPanel roboy.context.GUI.ContextGUI.valuePanel
Map<Context.Values, JLabel> roboy.context.GUI.ContextGUI.valueDisplays
Map<Context.ValueHistories, JScrollPane> roboy.context.GUI.ContextGUI.historyDisplays
TitledBorder roboy.context.GUI.ContextGUI.historyBorder
JPanel roboy.context.GUI.ContextGUI.historyPanel
JPanel roboy.context.GUI.ContextGUI.controlPanel
```

Private Static Attributes

```
int roboy.context.GUI.ContextGUI.MAX_HISTORY_VALUES = 10
int roboy.context.GUI.ContextGUI.FULL_WIDTH = 400
int roboy.context.GUI.ContextGUI.FULL_HEIGHT = 300
int roboy.context.GUI.ContextGUI.ATTR_WIDTH = 390
int roboy.context.GUI.ContextGUI.ATTR_HEIGHT = 50
int roboy.context.GUI.ContextGUI.HISTORY_HEIGHT = 100
String roboy.context.GUI.ContextGUI.NO_VALUE = "<not initialized>"
```

class ContextTest

Public Functions

```
void roboy.context.visionContext.ContextTest.getLastAttributeValue()
```

Checks that the values of FACE_COORDINATES get automatically updated.

```
void roboy.context.visionContext.ContextTest.setAndGetDialogTopics()
```

```
enum CONVERSATIONAL_STATE
```

Public Members

```
roboy.dialog.personality.DefaultPersonality.CONVERSATIONAL_STATE.GREETING
```

```
roboy.dialog.personality.DefaultPersonality.CONVERSATIONAL_STATE.INTRODUCTION
```

```
roboy.dialog.personality.DefaultPersonality.CONVERSATIONAL_STATE.SMALL_TALK
```

```
roboy.dialog.personality.DefaultPersonality.CONVERSATIONAL_STATE.FAREWELL
```

```
class ConverseState
```

Inherits from *roboy.dialog.personality.states.AbstractBooleanState*

Public Functions

```
roboy.dialog.personality.states.ConverseState.ConverseState()
```

```
List<Interpretation> roboy.dialog.personality.states.ConverseState.act()
```

```
Reaction roboy.dialog.personality.states.ConverseState.react(Interpretation input)
```

Protected Functions

```
boolean roboy.dialog.personality.states.ConverseState.determineSuccess(Interpretation input)
```

Private Members

```
State roboy.dialog.personality.states.ConverseState.inner
```

```
class CoordinateSet
```

A coordinate set data structure for the interlocutor face.

Public Functions

```
roboy.context.contextObjects.CoordinateSet.CoordinateSet(double x, double y, double z)
```

Package Attributes

```
final double roboy.context.contextObjects.CoordinateSet.x
```

```
final double roboy.context.contextObjects.CoordinateSet.y
```

```
final double roboy.context.contextObjects.CoordinateSet.z
```

class CuriousPersonality

Inherits from *robey.dialog.personality.Personality*

Public Functions

robey.dialog.personality.CuriousPersonality.CuriousPersonality()

List<Action> robey.dialog.personality.CuriousPersonality.answer(Interpretation input)
The central method of a personality.

Given an interpretation of all inputs (audio, visual, ...) to Roboy, this method decides which actions to perform in response.

Return A list of actions to perform in response

Parameters

- input: The interpretation of the inputs

Public Static Functions

static void robey.dialog.personality.CuriousPersonality.main(String[] args)

Private Functions

Triple robey.dialog.personality.CuriousPersonality.remember(String predicate, String a

Private Members

List<Triple> robey.dialog.personality.CuriousPersonality.memory

class DBpediaMemory

Restores information from the DBpedia.

Inherits from *robey.memory.Memory<Relation>*

Public Functions

boolean robey.memory.DBpediaMemory.save(Relation object)

List<Relation> robey.memory.DBpediaMemory.retrieve(Relation object)
Retrive all matching relations from DBpedia.

Public Static Functions

static DBpediaMemory robey.memory.DBpediaMemory.getInstance()
Loading DBpedia takes resources, so only do it once.

Return

static LinkedHashSet<String> robey.memory.DBpediaMemory.buildQueries(Relation object)

Private Functions

```
roboy.memory.DBpediaMemory.DBpediaMemory()
```

Private Members

```
Map<String,String> roboy.memory.DBpediaMemory.forms
```

Private Static Attributes

```
DBpediaMemory robby.memory.DBpediaMemory.dbpediaMemory
```

```
final Map<String, String> roboymemory.DBpediaMemory.supportedRelations
```

```
class DefaultPersonality
```

Inherits from *roboy.dialog.personality.Personality*

Public Functions

```
List<Action> roboy.dialog.personality.DefaultPersonality.answer(Interpretation input)
```

The central method of a personality.

Given an interpretation of all inputs (audio, visual, ...) to Roboy, this method decides which actions to perform in response.

Return A list of actions to perform in response

Parameters

- `input`: The interpretation of the inputs

Private Functions

```
String roboy.dialog.personality.DefaultPersonality.stripFromFront(String input, List<
```

```
String roboy.dialog.personality.DefaultPersonality.random(List< String > list)
```

```
boolean roboy.dialog.personality.DefaultPersonality.checkForTerm(String input, List< S
```

Private Members

```
CONVERSATIONAL_STATE robby.dialog.personality.DefaultPersonality.state = CONVERSATIONAL
```

Private Static Attributes

```
final List<String> roboy.dialog.personality.DefaultPersonality.greetings = Arrays.asList("hel
```

```
final List<String> roboy.dialog.personality.DefaultPersonality.farewells = Arrays.asList("cia")
```

```
final List<String> roboy.dialog.personality.DefaultPersonality.positive = Arrays.asList("enthu
```

```
final List<String> roboy.dialog.personality.DefaultPersonality.agreement = Arrays.asList("yes
```

```
final List<String> roboy.dialog.personality.DefaultPersonality.disagreement = Arrays.asList()
final List<String> roboy.dialog.personality.DefaultPersonality.introduction = Arrays.asList()
class DetectedEntity
```

Public Functions

```
roboy.linguistics.DetectedEntity.DetectedEntity(Entity entity, int tokenIndex)
Entity roboy.linguistics.DetectedEntity.getEntity()
int roboy.linguistics.DetectedEntity.getTokenIndex()
```

Private Members

```
Entity roboy.linguistics.DetectedEntity.entity
int roboy.linguistics.DetectedEntity.tokenIndex
```

class DialogStateMachine

State machine to manage dialog states.

Dialog state machines can be written to files and loaded from them later.

Personalities can be implemented using a dialog state machine.

Subclassed by *roboy.newDialog.StateBasedPersonality*

Public Functions

```
roboy.newDialog.DialogStateMachine.DialogStateMachine()
roboy.newDialog.DialogStateMachine.DialogStateMachine(boolean enableDebug)
State roboy.newDialog.DialogStateMachine.getInitialState()
void roboy.newDialog.DialogStateMachine.setInitialState(State initial)
    Set the initial state of this state machine.

    The state will be automatically added to the machine. If active state was null, it will be set to the new
    initial state.

Parameters
    • initial: initial state

void roboy.newDialog.DialogStateMachine.setInitialState(String identifier)
State roboy.newDialog.DialogStateMachine.getActiveState()
void roboy.newDialog.DialogStateMachine.setActiveState(State s)
void roboy.newDialog.DialogStateMachine.setActiveState(String identifier)
State roboy.newDialog.DialogStateMachine.getStateByIdentifier(String identifier)
void roboy.newDialog.DialogStateMachine.addState(State s)
void roboy.newDialog.DialogStateMachine.loadFromString(String s)
void roboy.newDialog.DialogStateMachine.loadFromFile(File f)
```



```

void robpy.newDialog.DialogStateMachine.saveToFile(File f)
String robpy.newDialog.DialogStateMachine.toJsonString()
String robpy.newDialog.DialogStateMachine.toString()
boolean robpy.newDialog.DialogStateMachine.equals(Object obj)

```

Private Functions

```

void robpy.newDialog.DialogStateMachine.loadFromJSON(JsonElement json)
JsonObject robpy.newDialog.DialogStateMachine.toJsonObject()

```

Private Members

```

HashMap<String, State> robpy.newDialog.DialogStateMachine.identifierToState
State robpy.newDialog.DialogStateMachine.activeState
State robpy.newDialog.DialogStateMachine.initalState
boolean robpy.newDialog.DialogStateMachine.enableDebug
class DialogStateMachineTest

```

Public Functions

```

void robpy.newDialog.DialogStateMachineTest.machineEqualsItself()
void robpy.newDialog.DialogStateMachineTest.stringEqualsCode()
void robpy.newDialog.DialogStateMachineTest.notEqualsNoInitialState()
void robpy.newDialog.DialogStateMachineTest.notEqualsDifferentStates()
void robpy.newDialog.DialogStateMachineTest.notEqualsDifferentTransitions()
void robpy.newDialog.DialogStateMachineTest.activeStateIsSetToInitialState()
void robpy.newDialog.DialogStateMachineTest.machineContainsAllStates()
void robpy.newDialog.DialogStateMachineTest.transitionsAreOK()
void robpy.newDialog.DialogStateMachineTest.fallbackIsOK()

```

Private Static Functions

```

static DialogStateMachine robpy.newDialog.DialogStateMachineTest.fromCode()

```

Private Static Attributes

```

String robpy.newDialog.DialogStateMachineTest.MINI_STATE_MACHINE = "{\n" + " " "initialState": "Greet

```

class DialogSystem

The dialog manager's main class.

Here, the used components are put together and executed using the main method. In the future, the different combinations of components should probably be transferred to configuration files.

The workflow in the dialog manager is the following:

1. Input devices produce an Input object
2. The Input object is transformed into an Interpretation object containing the input sentence in the Linguistics.SENTENCE attribute and all other lists of the Input object in the corresponding fields
3. Linguistic Analyzers are used on the Interpretation object to add additional information
4. The Personality class takes the Interpretation object and decides how to answer to this input
5. The list of actions returned by *Personality.answer* is performed by the Output devices
6. If one of these actions is a ShutDownAction the program terminates
7. Otherwise repeat

Input devices:

- For testing from command line: CommandLineInput
- For speech to text: BingInput (requires internet)
- For combining multiple inputs: MultiInputDevice
- Others for specific tasks

Analyzers:

- Tokenization: SimpleTokenizer
- Part-of-speech-tagging: OpenNLPPOSTagger
- Semantic role labeling: OpenNLPParser
- DBpedia question answering: AnswerAnalyzer
- Other more stupid ones

Personalities:

- SmallTalkPersonality: main one
- Others for testing specific things

Output devices:

- For testing with command line: CommandLineOutput
- For text to speech: BingOutput (requires internet)
- For combining multiple outputs: MultiOutputDevice
- For text to speech + facial expressions: CerevoiceOutput
- For facial expressions: EmotionOutput
- For text to speech (worse quality): FreeTTSTOutput

The easiest way to create ones own Roboy communication application is to pick the input and output devices provided here, use the tokenization, POS tagging and possibly semantic role labeling (though still under development) if needed and write an own personality. If one wants to use the DBpedia, Protege, generative model or state machine stuff, one has to dig deeper into the small talk personality and see how it is used there.

Public Static Functions

```
static void roboy.dialog.DialogSystem.main(String[] args)
```

class DialogTopics

Inherits from *roboy::context::ValueHistory< String >*

class DialogTopicsUpdater

Updater available to all DM for adding new values to the *DialogTopics* attribute.

Inherits from *roboy::context::InternalUpdater< DialogTopics, String >*

Public Functions

```
roboy.context.contextObjects.DialogTopicsUpdater.DialogTopicsUpdater(DialogTopics target)
```

class DictionaryBasedSentenceTypeDetector

Checks the sentence type by stupidly looking at the first word of the sentence and hoping that there is a known question word.

Puts the answer in the sentenceType variable of the *Interpretation* object.

Inherits from *roboy.linguistics.sentenceanalysis.Analyzer*

Public Functions

```
Interpretation roboy.linguistics.sentenceanalysis.DictionaryBasedSentenceTypeDetector.detect(Sentence sentence)
```

Private Functions

```
SENTENCE_TYPE roboy.linguistics.sentenceanalysis.DictionaryBasedSentenceTypeDetector.detect(Sentence sentence)
```

class DictionaryBasedSentenceTypeDetectorTest

Public Functions

```
void roboy.linguistics.sentenceanalysis.DictionaryBasedSentenceTypeDetectorTest.testWhisper()
```

Private Members

```
DictionaryBasedSentenceTypeDetector roboy.linguistics.sentenceanalysis.DictionaryBasedSentenceTypeDetector::instance
```

```
SimpleTokenizer roboy.linguistics.sentenceanalysis.DictionaryBasedSentenceTypeDetector::tokenizer
```

class DoubleMetaphoneEncoder

A phonetic encoder using the method double metaphone that maps words to their phonetic base form so that words that are written differently but sound similar receive the same form.

This is intended to be used to correct terms that Roboy misunderstood, but currently is not in use.

Inherits from *roboy.linguistics.phonetics.PhoneticEncoder*

Public Functions

```
roboy.linguistics.phonetics.DoubleMetaphoneEncoder.DoubleMetaphoneEncoder(DoubleMetaphon  
String roboy.linguistics.phonetics.DoubleMetaphoneEncoder.encode(String input)
```

Package Attributes

```
DoubleMetaphone roboy.linguistics.phonetics.DoubleMetaphoneEncoder.doubleMetaphone  
class EmotionAnalyzer  
Checks for a handfull of keywords and stores more or less fitting emotions in the Linguistics.EMOTION feature  
that is later read out and fed to the facial expression output module.  
Inherits from roboy.linguistics.sentenceanalysis.Analyzer
```

Public Functions

```
Interpretation roboy.linguistics.sentenceanalysis.EmotionAnalyzer.analyze(Interpretati  
class EmotionOutput  
Roboy's facial expression output.  
Inherits from roboy.io.OutputDevice
```

Public Functions

```
roboy.io.EmotionOutput.EmotionOutput(RosMainNode node)  
void roboy.io.EmotionOutput.act(List< Action > actions)  
void roboy.io.EmotionOutput.act(Action action)
```

Private Members

```
RosMainNode roboy.io.EmotionOutput.rosMainNode  
class Entity
```

Public Functions

```
roboy.linguistics.Entity.Entity(String term)  
String roboy.linguistics.Entity.getForm(String form)  
Map<String,String> roboy.linguistics.Entity.getForms()
```

Private Members

```
Map<String,String> roboy.linguistics.Entity.forms
```

interface ExternalContextInterface

Interface for an enum which lists *Context* values and valueHistories.

Methods enable retrieving values over generic methods with *AttributeManager*.

Subclassed by *roboy.context.Context.ValueHistories*, *roboy.context.Context.Values*

Public Functions

Class *roboy.context.ExternalContextInterface*.getClassType ()

Class *roboy.context.ExternalContextInterface*.getReturnType ()

class ExternalUpdater

For Values which should be updated upon incoming data or at regular intervals, this class fetches and passes the values.

Subclassed by *roboy.context.IntervalUpdater< T >*

Protected Functions

abstract void *roboy.context.ExternalUpdater*.update ()

class FaceAction

Action used if the dialogue manager wants Roboy to express a certain facial expression, like being angry, neutral or moving its lips (speak).

Inherits from *roboy.dialog.action.Action*

Public Functions

roboy.dialog.action.FaceAction.FaceAction(String state)

Constructor.

Duration is set to 1.

Parameters

- state: The facial expression. Possible values: angry, neutral, speak

roboy.dialog.action.FaceAction.FaceAction(String state, int duration)

Constructor.

Parameters

- state: The facial expression. Possible values: angry, neutral, speak
- duration: How long Roboy should display the given facial expression

String *roboy.dialog.action.FaceAction*.getState ()

int *roboy.dialog.action.FaceAction*.getDuration ()

Private Members

`String roboy.dialog.action.FaceAction.state`

`int roboy.dialog.action.FaceAction.duration`

class FaceCoordinates

xzy-coordinates of a person in the field of vision.

Inherits from *roboy::context::Value< CoordinateSet >*

class FaceCoordinatesUpdater

Asynchronously triggers ROS queries for face coordinates (in the future).

Inherits from *roboy::context::IntervalUpdater< FaceCoordinates >*

Public Functions

`roboy.context.contextObjects.FaceCoordinatesUpdater.FaceCoordinatesUpdater (FaceCoordinates)`

Protected Functions

`void roboy.context.contextObjects.FaceCoordinatesUpdater.update ()`

class FarewellState

Says goodbye.

Inherits from *roboy.dialog.personality.states.State*

Public Functions

`roboy.dialog.personality.states.FarewellState.FarewellState ()`

`List<Interpretation> roboy.dialog.personality.states.FarewellState.act ()`

`Reaction roboy.dialog.personality.states.FarewellState.react (Interpretation input)`

class FreeTTSOutput

Free TTS text to speech.

Inherits from *roboy.io.OutputDevice*

Public Functions

`roboy.io.FreeTTSOutput.FreeTTSOutput ()`

`void roboy.io.FreeTTSOutput.act (List< Action > actions)`

Public Static Functions

`static void roboy.io.FreeTTSOutput.main (String[] args)`

Private Members

Voice `roboy.io.FreeTTSOutput.voice`

class `GenerativeCommunicationState`

Inherits from *roboy.dialog.personality.states.AbstractBooleanState*

Public Functions

`List<Interpretation> roboy.dialog.personality.states.GenerativeCommunicationState.act()`

Protected Functions

`boolean roboy.dialog.personality.states.GenerativeCommunicationState.determineSuccess()`

Private Members

`boolean roboy.dialog.personality.states.GenerativeCommunicationState.first = true`

class `GreetingState`

Says hello.

Inherits from *roboy.dialog.personality.states.AbstractBooleanState*

Public Functions

`List<Interpretation> roboy.dialog.personality.states.GreetingState.act()`

`Reaction roboy.dialog.personality.states.GreetingState.react(Interpretation input)`

Protected Functions

`boolean roboy.dialog.personality.states.GreetingState.determineSuccess(Interpretation input)`

class `IdleState`

Inherits from *roboy.dialog.personality.states.AbstractBooleanState*

Public Functions

`List<Interpretation> roboy.dialog.personality.states.IdleState.act()`

Protected Functions

`boolean roboy.dialog.personality.states.IdleState.determineSuccess(Interpretation input)`

class `Input`

The result of an input device consists of a sentence, if it is an audio device, and an arbitrary map of lists.

Public Functions

```
roboy.io.Input.Input (String sentence)
roboy.io.Input.Input (String sentence, Map< String, Object > attributes)
```

Public Members

```
String roboy.io.Input.sentence
Map<String, Object> roboy.io.Input.attributes
```

interface InputDevice

An input device must listen and return an *Input* object.

Subclassed by *roboy.io.BingInput*, *roboy.io.CelebritySimilarityInput*, *roboy.io.CommandLineInput*, *roboy.io.MultiInputDevice*, *roboy.io.RoboyNameDetectionInput*, *roboy.io.UdpInput*

Public Functions

```
Input roboy.io.InputDevice.listen()
```

class InquiryState

Asks a given question, checks the answer for a list of given terms.

Moves to the success state if the answer consists one of these terms and to the failure state if not.

Inherits from *roboy.dialog.personality.states.AbstractBooleanState*

Public Functions

```
roboy.dialog.personality.states.InquiryState.InquiryState (String inquiry, List< String
Constructor.
```

Parameters

- *inquiry*: The question asked
- *successTerms*: The list of terms that is checked for
- *failureText*: Currently, not used

```
List<Interpretation> roboy.dialog.personality.states.InquiryState.act ()
```

Protected Functions

```
boolean roboy.dialog.personality.states.InquiryState.determineSuccess (Interpretation i
```

Private Members

```
String roboy.dialog.personality.states.InquiryState.inquiry
List<String> roboy.dialog.personality.states.InquiryState.successTerms
String roboy.dialog.personality.states.InquiryState.failureText
```


class IntentAnalyzer

Calls a machine learning model to determine if the utterance of the other person represents one of the learned intents.

Stores the highest scoring intent in the *Linguistics.INTENT* feature and the score in the *Linguistics.INTENT_DISTANCE* feature.

Inherits from *roboy.linguistics.sentenceanalysis.Analyzer*

Public Functions

```
roboy.linguistics.sentenceanalysis.IntentAnalyzer.IntentAnalyzer(RosMainNode ros)
```

```
Interpretation roboy.linguistics.sentenceanalysis.IntentAnalyzer.analyze(Interpretation)
```

Private Members

```
RosMainNode roboy.linguistics.sentenceanalysis.IntentAnalyzer.ros
```

interface Intention**class IntentionClassifier****Public Functions**

```
roboy.logic.IntentionClassifier.IntentionClassifier(Ros ros_)
```

```
String roboy.logic.IntentionClassifier.classify(String utterance)
```

Private Members

```
Ros roboy.logic.IntentionClassifier.ros
```

class Interlocutor

Encapsulates a *MemoryNodeModel* and enables dialog states to easily store and retrieve information about its current conversation partner.

Public Functions

```
roboy.memory.nodes.Interlocutor.Interlocutor()
```

```
void roboy.memory.nodes.Interlocutor.addName(String name)
```

After executing this method, the person field contains a node that is in sync with memory and represents the interlocutor.

Unless something goes wrong during querying, which would affect the following communication severely.

```
String roboy.memory.nodes.Interlocutor.getName()
```

```
boolean roboy.memory.nodes.Interlocutor.hasRelationship(Neo4jRelationships type)
```

```
ArrayList<Integer> roboy.memory.nodes.Interlocutor.getRelationships(Neo4jRelationships type)
```

```
void roboy.memory.nodes.Interlocutor.addInformation(String relationship, String name)
```

Adds a new relation to the person node, updating memory.

Public Members

```
boolean roboy.memory.nodes.Interlocutor.FAMILIAR = false
```

Package Attributes

```
Neo4jMemory roboy.memory.nodes.Interlocutor.memory
```

Private Members

```
MemoryNodeModel roboy.memory.nodes.Interlocutor.person
```

```
boolean roboy.memory.nodes.Interlocutor.memoryROS
```

```
template <T, V>
```

```
class InternalUpdater
```

An updater which can be called from inside DM to update a *Value* or *ValueHistory*.

Parameters

- <T>: The target *Value* or *ValueHistory*.
- <V>: The data type stored in the target.

Subclassed by *roboy.context.contextObjects.DialogTopicsUpdater*

Public Functions

```
synchronized void roboy.context.InternalUpdater< T extends AbstractValue< V, V >.putVa
```

Protected Functions

```
roboy.context.InternalUpdater< T extends AbstractValue< V, V >.InternalUpdater(T target
```

Package Attributes

```
AbstractValue<V> roboy.context.InternalUpdater< T extends AbstractValue< V, V >.target
```

```
class Interpretation
```

An interpretation of all inputs to Roboy consists of the sentence type and an arbitrary map of features.

Feature names are listed and documented in the class *roboy.linguistics.Linguistics*.

The interpretation class is also used to pass the output information from the states to the verbalizer class.

Public Functions

```
roboy.linguistics.sentenceanalysis.Interpretation.Interpretation(String sentence)
```

```
roboy.linguistics.sentenceanalysis.Interpretation.Interpretation(String sentence, Map<
```

```
roboy.linguistics.sentenceanalysis.Interpretation.Interpretation(SENTENCE_TYPE sentence
```

```

roboy.linguistics.sentenceanalysis.Interpretation.Interpretation(SENTENCE_TYPE sentenceType)
Map<String, Object> roboy.linguistics.sentenceanalysis.Interpretation.getFeatures()
Object roboy.linguistics.sentenceanalysis.Interpretation.getFeature(String featureName)
void roboy.linguistics.sentenceanalysis.Interpretation.setFeatures(Map<String, Object> features)
SENTENCE_TYPE roboy.linguistics.sentenceanalysis.Interpretation.getSentenceType()
void roboy.linguistics.sentenceanalysis.Interpretation.setSentenceType(SENTENCE_TYPE sentenceType)
String roboy.linguistics.sentenceanalysis.Interpretation.toString()

```

Private Members

```

Map<String, Object> roboy.linguistics.sentenceanalysis.Interpretation.features
SENTENCE_TYPE roboy.linguistics.sentenceanalysis.Interpretation.sentenceType
template <T>
class IntervalUpdater

```

An implementation of the UpdatePolicy which performs regular updates on a target object.

The method update() needs to be implemented in the subclass.

Parameters

- <T>: The class of the target object.

Inherits from *roboy.context.ExternalUpdater*

Subclassed by *roboy.context.contextObjects.FaceCoordinatesUpdater*

Public Functions

```

roboy.context.IntervalUpdater< T >.IntervalUpdater(T target, int updateFrequencySeconds)

```

Create a new updater service, executing the update() method at regular time intervals.

Parameters

- target: The target attribute of the update() method.
- updateFrequencySeconds: Delay in seconds between calls to the update() method.

Public Members

```

final int roboy.context.IntervalUpdater< T >.updateFrequency

```

Protected Attributes

```

final T roboy.context.IntervalUpdater< T >.target
final ScheduledExecutorService roboy.context.IntervalUpdater< T >.scheduler = Executors.newSingleThreadExecutor()

```

Private Functions

```
void roboy.context.IntervalUpdater< T >.start ()
```

Starts the ScheduledExecutorService of the updating thread.

class IntroductionState

Roboy introduces himself and asks “Who are you?”.

Moves to success state if the answer is at most 2 words.

Inherits from *roboy.dialog.personality.states.AbstractBooleanState*

Public Functions

```
roboy.dialog.personality.states.IntroductionState.IntroductionState (Interlocutor person)
```

```
List<Interpretation> roboy.dialog.personality.states.IntroductionState.act ()
```

Public Members

```
Neo4jRelationships roboy.dialog.personality.states.IntroductionState.predicate = Neo4jRel
```

Protected Functions

```
boolean roboy.dialog.personality.states.IntroductionState.determineSuccess (Interpretation)
```

Performs person detection by consulting memory.

Package Attributes

```
Interlocutor roboy.dialog.personality.states.IntroductionState.person = new Interlocutor()
```

```
Neo4jMemory roboy.dialog.personality.states.IntroductionState.memory
```

Private Static Attributes

```
final List<String> roboy.dialog.personality.states.IntroductionState.introductions = List
```

class IO

Helper class for *IO* related tasks.

Public Static Functions

```
static String roboy.util.IO.readFile (String file)
```

```
static String roboy.util.IO.readFile (File file)
```

```
static List<String> roboy.util.IO.readLines (String file)
```

```
static List<String> roboy.util.IO.readLines (File file)
```

class JsonEntryModel

Package Attributes

```

List<String> roboy.util.JsonEntryModel.Q
Map<String, List<String> > roboy.util.JsonEntryModel.A
Map<String, List<String> > roboy.util.JsonEntryModel.FUP
class JsonModel

```

Package Attributes

```

JsonEntryModel roboy.util.JsonModel.name
JsonEntryModel roboy.util.JsonModel.FROM
JsonEntryModel roboy.util.JsonModel.HAS_HOBBY
JsonEntryModel roboy.util.JsonModel.LIVE_IN
JsonEntryModel roboy.util.JsonModel.FRIEND_OF
JsonEntryModel roboy.util.JsonModel.STUDY_AT
JsonEntryModel roboy.util.JsonModel.MEMBER_OF
JsonEntryModel roboy.util.JsonModel.WORK_FOR
JsonEntryModel roboy.util.JsonModel.OCCUPIED_AS
class JsonQAValues

```

Public Functions

```

roboy.util.JsonQAValues.JsonQAValues(Map< String, List< String >> questions, Map< Strin
Map<String, List<String> > roboy.util.JsonQAValues.getQuestions()
Map<String, List<String> > roboy.util.JsonQAValues.getSuccessAnswers()
Map<String, List<String> > roboy.util.JsonQAValues.getFailureAnswers()
Map<String, List<String> > roboy.util.JsonQAValues.getFollowUpQuestions()
Map<String, List<String> > roboy.util.JsonQAValues.getFollowUpAnswers()

```

Private Members

```

Map<String, List<String> > roboy.util.JsonQAValues.questions
Map<String, List<String> > roboy.util.JsonQAValues.successAnswers
Map<String, List<String> > roboy.util.JsonQAValues.failureAnswers
Map<String, List<String> > roboy.util.JsonQAValues.followUp
Map<String, List<String> > roboy.util.JsonQAValues.answersFollowUp
class JsonUtils

```

Public Static Functions

static JsonQAValues roboy.util.JsonUtils.getQuestionsAndAnswersFromJson(String file)
Fetches the complete JSON string, splits and converts the most straightforward way into backward-compatible Map<> entries initializing a backward-compatible *JsonQAValues* class.

static Map<String, List<String> > roboy.util.JsonUtils.getSentencesFromJsonFile(String filename)
Fetches the map of (keyword) -> (lists of corresponding questions) from the specified filename.

static Map<String, List<String[]> > roboy.util.JsonUtils.getSentenceArraysFromJsonFile(String filename)
The success responses consist of an array of two strings, which enables reflecting parsed answers back at the conversation partner.

class KnockKnockPersonality

A test personality that only tries to tell knock knock jokes.

This should later be included in sensible states and used in the normal small talk personality.

Inherits from *roboy.dialog.personality.Personality*

Public Functions

List<Action> roboy.dialog.personality.KnockKnockPersonality.answer(Interpretation input)
The personality has four states which it will run through consecutively: WELCOME: An initial greeting, starting with a knock, knock KNOCKKNOCK: Knock, knock after at least one joke was told already WHOISTHERE: Giving the first, shortened answer PUNCHLINE: Giving the punchline of the joke.

Private Functions

String [] roboy.dialog.personality.KnockKnockPersonality.pickJoke()

Private Members

KnockKnockState roboy.dialog.personality.KnockKnockPersonality.state = KnockKnockState.WELCOME

String [] roboy.dialog.personality.KnockKnockPersonality.joke

String [][] roboy.dialog.personality.KnockKnockPersonality.jokes = new String[][]{ new String[][]{“

enum KnockKnockState

Public Members

roboy.dialog.personality.KnockKnockPersonality.KnockKnockState.WELCOME

roboy.dialog.personality.KnockKnockPersonality.KnockKnockState.KNOCKKNOCK

roboy.dialog.personality.KnockKnockPersonality.KnockKnockState.WHOSETHERE

roboy.dialog.personality.KnockKnockPersonality.KnockKnockState.PUNCHLINE

class Lexicon

Represents a Protege lexicon.

Public Functions

```

roboy.memory.Lexicon.Lexicon()
List<LexiconLiteral> roboy.memory.Lexicon.getLiterals(String question, int limit, int ...)
List<LexiconPredicate> roboy.memory.Lexicon.scoreThesePredicates(List< LexiconPredicate> list)
List<LexiconLiteral> roboy.memory.Lexicon.addTypeOfOwner(List< LexiconLiteral > result)
List<LexiconLiteral> roboy.memory.Lexicon.scoreLiterals(List< LexiconLiteral > result)
List<String> roboy.memory.Lexicon.getPermutations(String question)

```

Package Functions

```
String roboy.memory.Lexicon.bestLabelOf(String objlabel, String label1, String permutation)
```

Private Functions

```
List<LexiconPredicate> roboy.memory.Lexicon.addDomainAndRange(List< LexiconPredicate> list)
```

Private Members

```

List<LexiconPredicate> roboy.memory.Lexicon.predicateList
List<LexiconLiteral> roboy.memory.Lexicon.literalList
Boolean roboy.memory.Lexicon.predicateFilled
Boolean roboy.memory.Lexicon.literalFilled
List<String> roboy.memory.Lexicon.permutationList

```

class LexiconLiteral

An entity in the lexicon.

Inherits from Comparable<LexiconLiteral>

Public Functions

```

roboy.memory.LexiconLiteral.LexiconLiteral()
roboy.memory.LexiconLiteral.LexiconLiteral(String URI, String label, String QuestionMark)
roboy.memory.LexiconLiteral.LexiconLiteral(String URI, String label, String QuestionMark)
int roboy.memory.LexiconLiteral.compareTo(LexiconLiteral lexlit)

```

Public Members

```

List<String> roboy.memory.LexiconLiteral.typeOfOwner
String roboy.memory.LexiconLiteral.URI
String roboy.memory.LexiconLiteral.label

```

```
String roboy.memory.LexiconLiteral.QuestionMatch
int roboy.memory.LexiconLiteral.score
```

Public Static Attributes

```
public int compare(LexiconLiteral lexlit1, LexiconLiteral lexlit2) { return lexlit1.compareTo(lexlit2); } }]
```

class LexiconPredicate
A relation in the lexicon.
Inherits from Comparable< LexiconPredicate >

Public Functions

```
roboy.memory.LexiconPredicate.LexiconPredicate()
roboy.memory.LexiconPredicate.LexiconPredicate(String URI, String Label)
int roboy.memory.LexiconPredicate.compareTo(LexiconPredicate lexpre)
```

Public Members

```
List<String> roboy.memory.LexiconPredicate.domains
List<String> roboy.memory.LexiconPredicate.ranges
String roboy.memory.LexiconPredicate.type
String roboy.memory.LexiconPredicate.URI
String roboy.memory.LexiconPredicate.label
String roboy.memory.LexiconPredicate.QuestionMatch
int roboy.memory.LexiconPredicate.score
```

Public Static Attributes

```
return lexpre1.compareTo(lexpre2); } }]
```

class Linguistics
Collection of attribute names, enumerations, word lists etc.
related to linguistics.

Most importantly it contains the names of the results of the Analyzer that are stored in an Interpretation object and can be retrieved by the `getFeature(String featureName)` method. These feature names include: SENTENCE TRIPLE TOKENS POSTAGS KEYWORDS ASSOCIATION PAS NAME CELEBRITY OBJ_ANSWER PRED_ANSWER EMOTION INTENT INTENT_DISTANCE

Public Static Attributes

```
final List<String> roboy.linguistics.Linguistics.tobe = Arrays.asList("am","are","is","was","were","be")
final List<String> roboy.linguistics.Linguistics.beMod = Lists.stringList("am","are","is","was","were",
```



```

final String robby.linguistics.Linguistics.SENTENCE = "sentence"
    The utterance of the person Roboy is speaking to.

final String robby.linguistics.Linguistics.TRIPLE = "triple"
    A triple of subject, predicate and object extracted by a very primitive rule system.

final String robby.linguistics.Linguistics.TOKENS = "tokens"
    The tokens (usually words) of the sentence.

final String robby.linguistics.Linguistics.POSTAGS = "postags"
    The part-of-speech tags (noun, verb, adjective etc.) corresponding to the tokens.

final String robby.linguistics.Linguistics.KEYWORDS = "keywords"
    If keywords for the segue state from the resource knowledgebase/triviaWords.csv are detected, they are
    passed with this name.

final String robby.linguistics.Linguistics.ASSOCIATION = "association"
    Is used to pass the detected keyword from the segue state to the verbalizer state to mention it before telling
    the anecdote.

final String robby.linguistics.Linguistics.PAS = "pas"
    Predicate-argument structures (who(agens) did what(predicate) to whom(patients))

final String robby.linguistics.Linguistics.NAME = "name"
    Internally used to retrieve the name of a concept.

final String robby.linguistics.Linguistics.CELEBRITY = "celebrity"
    The name of the celebrity most resembling the person talked to, as detected by the CelebritySimilarityInput.

final String robby.linguistics.Linguistics.ROBOYDETECTED = "roboydetected"
    If Roboy detected his own name.

final String robby.linguistics.Linguistics.OBJ_ANSWER = "objanswer"
    Contains the answer to a question asked by the QuestionAskingState, if the answer is expected to be in the
    object of the sentence, like if the question is "What is your name?" or "Where are you from?".

final String robby.linguistics.Linguistics.PRED_ANSWER = "predanswer"
    Contains the answer to a question asked by the QuestionAskingState, if the answer is expected to be a
    predicate or a predicate and an object of the sentence, like if the question is "What is your hobby?" or
    "What do you do for a living?".

final String robby.linguistics.Linguistics.EMOTION = "emotion"
    Contains the emotion Roboy intends to express based on the keyword detection in the EmotionAnalyzer.

final String robby.linguistics.Linguistics.INTENT = "intent"
    The result of the machine learning intent classification in the IntentAnalyzer.

final String robby.linguistics.Linguistics.INTENT_DISTANCE = "intentdistance"
    The confidence score of the machine learning intent classification in the IntentAnalyzer.

final String robby.linguistics.Linguistics.PARSE = "parse"
    The result of SemanticParserAnalyzer, formal language representation.

```

class Lists

Helper class for list related tasks.

Public Static Functions

```

static List<Action> robby.util.Lists.actionList (Action... actions)

```

```
static List<Interpretation> roboy.util.Lists.interpretationList (Interpretation... inte
static List<String> roboy.util.Lists.stringList (String... strings)
static List<String[]> roboy.util.Lists.strArray (String... [] strings)
```

class LocationDBpedia

Created by roboy on 7/5/17.

Inherits from *roboy.dialog.personality.states.AbstractBooleanState*

Public Functions

```
List<Interpretation> roboy.dialog.personality.states.LocationDBpedia.act ()
boolean roboy.dialog.personality.states.LocationDBpedia.determineSuccess (Interpretatio
Reaction roboy.dialog.personality.states.LocationDBpedia.react (Interpretation input)
```

class LocationDBpediaStateTest

Created by roboy on 7/5/17.

Public Functions

```
void roboy.dialog.personality.states.LocationDBpediaStateTest.testCity ()
void roboy.dialog.personality.states.LocationDBpediaStateTest.testCountry ()
```

class Maps

Helper class for map related tasks.

Public Static Functions

```
static Map<String, String> roboy.util.Maps.stringMap (String... elements)
static Map<String, Object> roboy.util.Maps.stringObjectMap (Object... elements)
static Map<String, Reaction> roboy.util.Maps.stringReactionMap (Object... elements)
static Map<Integer, String> roboy.util.Maps.intStringMap (Object... elements)
template <T>
interface Memory
```

The *Memory* interface contains of methods to save and retrieve information.

Parameters

- <T>: the type of information stored

Public Functions

```
boolean roboy.memory.Memory< T >.save (T object)
```

Storing the element in the memory.

Return true, if storing was successful

Parameters

- `object`: the element to be stored

Exceptions

- `InterruptedException`:
- `IOException`:

List<T> roboy.memory.Memory< T >.retrieve(T object)

Retrieve an element from memory.

Return a list of objects that match the query containing all the required information

Parameters

- `object`: a version of the object that lacks information (e.g. it only has the ID)

Exceptions

- `InterruptedException`:
- `IOException`:

class MemoryNodeModel

This class represents a full node similarly to its representation in *Memory*.

Public Functions

`roboy.memory.nodes.MemoryNodeModel.MemoryNodeModel()`

`roboy.memory.nodes.MemoryNodeModel.MemoryNodeModel(boolean stripQuery)`

`int roboy.memory.nodes.MemoryNodeModel.getId()`

`void roboy.memory.nodes.MemoryNodeModel.setId(int id)`

`ArrayList<String> roboy.memory.nodes.MemoryNodeModel.getLabels()`

`void roboy.memory.nodes.MemoryNodeModel.setLabel(String label)`

`HashMap<String, Object> roboy.memory.nodes.MemoryNodeModel.getProperties()`

`Object roboy.memory.nodes.MemoryNodeModel.getProperty(String key)`

`void roboy.memory.nodes.MemoryNodeModel.setProperties(HashMap<String, Object> properties)`

`void roboy.memory.nodes.MemoryNodeModel.setProperty(String key, Object property)`

`HashMap<String, ArrayList<Integer>> roboy.memory.nodes.MemoryNodeModel.getRelationships()`

`ArrayList<Integer> roboy.memory.nodes.MemoryNodeModel.getRelationship(String key)`

`void roboy.memory.nodes.MemoryNodeModel.setRelationships(HashMap<String, ArrayList<Integer>> relationships)`

`void roboy.memory.nodes.MemoryNodeModel.setRelationship(String key, Integer id)`

`void roboy.memory.nodes.MemoryNodeModel.setStripQuery(boolean strip)`

`String roboy.memory.nodes.MemoryNodeModel.toJSON(Gson gson)`

This toString method returns the whole object, including empty variables.

`MemoryNodeModel roboy.memory.nodes.MemoryNodeModel.fromJSON(String json, Gson gson)`

Returns an instance of this class based on the given JSON.

Package Attributes

```
transient boolean roboy.memory.nodes.MemoryNodeModel.stripQuery = false
```

Private Members

```
int roboy.memory.nodes.MemoryNodeModel.id
ArrayList<String> roboy.memory.nodes.MemoryNodeModel.labels
String roboy.memory.nodes.MemoryNodeModel.label
HashMap<String, Object> roboy.memory.nodes.MemoryNodeModel.properties
HashMap<String, ArrayList<Integer> > roboy.memory.nodes.MemoryNodeModel.relationships
```

class MetaphoneEncoder

A phonetic encoder using the method metaphone that maps words to their phonetic base form so that words that are written differently but sound similar receive the same form.

This is intended to be used to correct terms that Roboy misunderstood, but currently is not is use.

Inherits from *roboy.linguistics.phonetics.PhoneticEncoder*

Public Functions

```
roboy.linguistics.phonetics.MetaphoneEncoder.MetaphoneEncoder (Metaphone metaphone)
String roboy.linguistics.phonetics.MetaphoneEncoder.encode (String input)
```

Private Members

```
Metaphone roboy.linguistics.phonetics.MetaphoneEncoder.metaphone
```

class MultiInputDevice

Meta class to combine multiple input devices.

Inherits from *roboy.io.InputDevice*

Public Functions

```
roboy.io.MultiInputDevice.MultiInputDevice (InputDevice mainInput)
void roboy.io.MultiInputDevice.addInputDevice (InputDevice additionalInput)
Input roboy.io.MultiInputDevice.listen ()
```

Private Members

```
InputDevice roboy.io.MultiInputDevice.mainInput
ArrayList<InputDevice> roboy.io.MultiInputDevice.additionalInputs
```

class MultiOutputDevice

Meta class to combine multiple output devices.

Inherits from *robey.io.OutputDevice*

Public Functions

```
robey.io.MultiOutputDevice.MultiOutputDevice(OutputDevice device)
void robey.io.MultiOutputDevice.add(OutputDevice additionalDevice)
void robey.io.MultiOutputDevice.act(List< Action > actions)
```

Private Members

```
ArrayList<OutputDevice> robey.io.MultiOutputDevice.devices
```

class Neo4jMemory

Implements the high-level-querying tasks to the *Memory* services using RosMainNode.

Inherits from *robey.memory.Memory< MemoryNodeModel >*

Public Functions

```
boolean robey.memory.Neo4jMemory.save(MemoryNodeModel node)
    Updating information in the memory for an EXISTING node with known ID.
```

Return true for success, false for fail

Parameters

- node: Node with a set ID, and other properties to be set or updated.

```
MemoryNodeModel robey.memory.Neo4jMemory.getById(int id)
    This query retrieves a a single node by its ID.
```

Return Node representation of the result.

Parameters

- id: the ID of requested

```
ArrayList<Integer> robey.memory.Neo4jMemory.getByQuery(MemoryNodeModel query)
    This is a classical database query which finds all matching nodes.
```

Return Array of IDs (all nodes which correspond to the pattern).

Parameters

- query: the ID of requested

```
int robey.memory.Neo4jMemory.create(MemoryNodeModel query)
```

```
boolean robey.memory.Neo4jMemory.remove(MemoryNodeModel query)
    IF ONLY THE ID IS SET, THE NODE IN MEMORY WILL BE DELETED ENTIRELY.
```

Otherwise, the properties present in the query will be deleted.

Parameters

- query: StrippedQuery avoids accidentally deleting other fields than intended.

```
List<MemoryNodeModel> roboy.memory.Neo4jMemory.retrieve(MemoryNodeModel query)
//TODO Deprecated due to interface incompatibility, use getById or getByMatch
```

Return Array with a single node

Parameters

- query: a GetByIdQuery instance

```
String roboy.memory.Neo4jMemory.determineNodeType(String relationship)
```

Public Static Functions

```
static Neo4jMemory roboy.memory.Neo4jMemory.getInstance(RosMainNode node)
static Neo4jMemory roboy.memory.Neo4jMemory.getInstance()
```

Private Functions

```
roboy.memory.Neo4jMemory.Neo4jMemory(RosMainNode node)
```

Private Members

```
Gson roboy.memory.Neo4jMemory.gson = new Gson()
```

Private Static Attributes

```
Neo4jMemory roboy.memory.Neo4jMemory.memory
RosMainNode roboy.memory.Neo4jMemory.rosMainNode
```

enum Neo4jRelationships

Contains the relations available in Neo4j database.

Respective questions should be added to the questions.json file and used in the QuestionRandomizerState.

Public Functions

```
roboy.memory.Neo4jRelationships.Neo4jRelationships(String type)
```

Public Members

```
roboy.memory.Neo4jRelationships.FROM = ("FROM")
roboy.memory.Neo4jRelationships.HAS_HOBBY = ("HAS_HOBBY")
roboy.memory.Neo4jRelationships.LIVE_IN = ("LIVE_IN")
roboy.memory.Neo4jRelationships.STUDY_AT = ("STUDY_AT")
roboy.memory.Neo4jRelationships.OCCUPIED_AS = ("OCCUPIED_AS")
```

```

robey.memory.Neo4jRelationships.WORK_FOR=("WORK_FOR")
robey.memory.Neo4jRelationships.FRIEND_OF=("FRIEND_OF")
robey.memory.Neo4jRelationships.MEMBER_OF=("MEMBER_OF")
robey.memory.Neo4jRelationships.OTHER=("OTHER")
robey.memory.Neo4jRelationships.IS=("IS")
String robey.memory.Neo4jRelationships.type

```

class NewDialogSystem

Temporary class to test new state based personality.

Will be extended and might replace the old DialogSystem in the future.

Public Static Functions

```
static void robey.newDialog.NewDialogSystem.main(String[] args)
```

Private Static Functions

```
static String robey.newDialog.NewDialogSystem.getPersonalityFilePathFromConfig()
```

class OntologyNERAnalyzer

Checks for keywords from a list (knowledgebase/triviaWords.csv) and stores them in *Linguistics.KEYWORDS* attribute of the interpretation.

Inherits from *robey.linguistics.sentenceanalysis.Analyzer*

Public Functions

```
robey.linguistics.sentenceanalysis.OntologyNERAnalyzer.OntologyNERAnalyzer()
```

```
Interpretation robey.linguistics.sentenceanalysis.OntologyNERAnalyzer.analyze(Interpre
```

Private Members

```
Map<String,Entity> robey.linguistics.sentenceanalysis.OntologyNERAnalyzer.entities
```

class OpenNLPParser

Performs a sentence analysis using the Open NLP constituency parser, then interprets the output for predicate argument structures (who did what to whom?) and stores them in the *Linguistics.PAS* attribute of the interpretation.

Inherits from *robey.linguistics.sentenceanalysis.Analyzer*

Public Functions

```
robey.linguistics.sentenceanalysis.OpenNLPParser.OpenNLPParser()
```

```
Interpretation robey.linguistics.sentenceanalysis.OpenNLPParser.analyze(Interpretation
```

```
StringBuilder robey.linguistics.sentenceanalysis.OpenNLPParser.parseToString(Parse par
```

Public Static Functions

```
static void robey.linguistics.sentenceanalysis.OpenNLPParser.main(String[] args)
```

Private Functions

```
Interpretation robey.linguistics.sentenceanalysis.OpenNLPParser.extractPAS(Interpretation)
Map<SEMANTIC_ROLE, Object> robey.linguistics.sentenceanalysis.OpenNLPParser.top(Parse p)
Map<SEMANTIC_ROLE, Object> robey.linguistics.sentenceanalysis.OpenNLPParser.sbar(Parse p)
Map<SEMANTIC_ROLE, Object> robey.linguistics.sentenceanalysis.OpenNLPParser.vp(Parse p)
```

Private Members

```
Parser robey.linguistics.sentenceanalysis.OpenNLPParser.parser
class OpenNLPParserTest
```

Public Functions

```
void robey.linguistics.sentenceanalysis.OpenNLPParserTest.testWhatIs()
void robey.linguistics.sentenceanalysis.OpenNLPParserTest.testWhenWas()
void robey.linguistics.sentenceanalysis.OpenNLPParserTest.testWhereWas()
void robey.linguistics.sentenceanalysis.OpenNLPParserTest.testWhereDid()
void robey.linguistics.sentenceanalysis.OpenNLPParserTest.testWhenDid()
void robey.linguistics.sentenceanalysis.OpenNLPParserTest.testHowAdjective()
```

Private Static Attributes

```
final OpenNLPParser robey.linguistics.sentenceanalysis.OpenNLPParserTest.parser = new OpenNLPParser()
class OpenNLPPPOSTagger
    Perform part-of-speech tagging (detecting nouns, verbs etc.) using the Open NLP POS tagger and stores the
    results in the Linguistics.POSTAGS attribute of the interpretation.
    Inherits from robey.linguistics.sentenceanalysis.Analyzer
```

Public Functions

```
robey.linguistics.sentenceanalysis.OpenNLPPPOSTagger.OpenNLPPPOSTagger()
Interpretation robey.linguistics.sentenceanalysis.OpenNLPPPOSTagger.analyze(Interpretation)
```

Private Functions

```
String [] robey.linguistics.sentenceanalysis.OpenNLPPPOSTagger.postTag(String[] tokens)
```


Private Members

POSTaggerME `roboy.linguistics.sentenceanalysis.OpenNLPPPOSTagger.tagger`

interface `OutputDevice`

An output device gets a list of actions and should perform those that it can handle.

Subclassed by `roboy.io.BingOutput`, `roboy.io.CerevoiceOutput`, `roboy.io.CommandLineOutput`, `roboy.io.EmotionOutput`, `roboy.io.FreeTTSOutput`, `roboy.io.MultiOutputDevice`, `roboy.io.UdpOutput`

Public Functions

void `roboy.io.OutputDevice.act(List< Action > actions)`

class `PASInterpreter`

Turns predicate-argument-structures in the kind of relations required for querying DBpedia.

Maps the predicates to the predicate keys of DBpedia and picks the elements of the relation from different arguments of the PAS depending on the relation type.

Public Static Functions

static Relation `roboy.logic.PASInterpreter.pas2DBpediaRelation(Map< String, Object > p`

Transforms a predicate argument structure into a DBpedia relation, that can be used to query DBpedia for the answer to the missing elements of the PAS.

Return the DBpedia relation

Parameters

- `pas`: the predicate argument structure

Private Static Attributes

final Map<String, String> `roboy.logic.PASInterpreter.dbpediaRelations`

class `PASInterpreterTest`

Public Functions

void `roboy.logic.PASInterpreterTest.testWhenWas()`

void `roboy.logic.PASInterpreterTest.testWhatIs()`

void `roboy.logic.PASInterpreterTest.testWhereWas()`

void `roboy.logic.PASInterpreterTest.testWhereDid()`

void `roboy.logic.PASInterpreterTest.testWhenDid()`

void `roboy.logic.PASInterpreterTest.testHowAdjective()`

void `roboy.logic.PASInterpreterTest.testWhatIsNewExamples()`

void `roboy.logic.PASInterpreterTest.testWhereDidNewExamples()`

void `roboy.logic.PASInterpreterTest.testWhoIsNewExamples()`

```
void roboy.logic.PASInterpreterTest.testWhoLivesNewExamples ()
```

Private Static Attributes

```
final OpenNLPParser roboy.logic.PASInterpreterTest.parser = new OpenNLPParser()
```

```
class PersistentKnowledge
```

CSV file memory.

Can only be used for retrieving and not for storing.

Inherits from *roboy.memory.Memory< Triple >*

Public Functions

```
List<Triple> roboy.memory.PersistentKnowledge.retrieve(Triple triple)
```

```
boolean roboy.memory.PersistentKnowledge.save(Triple triple)
```

Public Static Functions

```
static PersistentKnowledge roboy.memory.PersistentKnowledge.getInstance ()
```

Private Functions

```
roboy.memory.PersistentKnowledge.PersistentKnowledge ()
```

Private Static Attributes

```
PersistentKnowledge roboy.memory.PersistentKnowledge.persistentKnowledge
```

```
List<Triple> roboy.memory.PersistentKnowledge.memory
```

```
class PersonalFollowUpState
```

Inherits from *roboy.dialog.personality.states.AbstractBooleanState*

Public Functions

```
roboy.dialog.personality.states.PersonalFollowUpState.PersonalFollowUpState(List< Stri
```

```
List<Interpretation> roboy.dialog.personality.states.PersonalFollowUpState.act ()
```

Ask the question.

Using Neo4jRelationships predicate

Public Members

```
Neo4jRelationships roboy.dialog.personality.states.PersonalFollowUpState.predicate
```

Protected Functions

boolean `robey.dialog.personality.states.PersonalFollowUpState.determineSuccess` (Interpretation input)
Retrieve the answer and add it to the memory, if needed.

As locations, hobbies, workplaces etc are individual nodes in memory, those will be retrieved or created if necessary.

Private Members

List<String> `robey.dialog.personality.states.PersonalFollowUpState.questions`

List<String> `robey.dialog.personality.states.PersonalFollowUpState.successTexts`

Interlocutor `robey.dialog.personality.states.PersonalFollowUpState.person`

interface `Personality`

Personality interface.

A personality is designed to define how Roboy reacts in every given situation. Roboy can always only represent one personality at a time. Different personalities are meant to be used in different situations, like a more formal or loose one depending on the occasion where he is at. In the future, also different languages could be realized by the use of different personalities.

The currently used personality is the *SmallTalkPersonality* which makes use of a state machine to act and react differently in different situations.

Subclassed by *robey.dialog.personality.CuriousPersonality*, *robey.dialog.personality.DefaultPersonality*, *robey.dialog.personality.KnockKnockPersonality*, *robey.dialog.personality.SmallTalkPersonality*, *robey.newDialog.StateBasedPersonality*

Public Functions

List<Action> `robey.dialog.personality.Personality.answer` (Interpretation input)

The central method of a personality.

Given an interpretation of all inputs (audio, visual, ...) to Roboy, this method decides which actions to perform in response.

Return A list of actions to perform in response

Parameters

- `input`: The interpretation of the inputs

class `PersonalQASState`

Inherits from *robey.dialog.personality.states.AbstractBooleanState*

Public Functions

robey.dialog.personality.states.PersonalQASState.PersonalQASState (List< String > question)

List<Interpretation> `robey.dialog.personality.states.PersonalQASState.act` ()

Ask the question.

Public Members

`Neo4jRelationships roboy.dialog.personality.states.PersonalQAState.predicate`

Protected Functions

`boolean roboy.dialog.personality.states.PersonalQAState.determineSuccess (Interpretation)`
Retrieve the answer and add it to the memory, if needed.

As locations, hobbies, workplaces etc are individual nodes in memory, those will be retrieved or created if necessary.

Private Members

`List<String> roboy.dialog.personality.states.PersonalQAState.questions`

`List<String> roboy.dialog.personality.states.PersonalQAState.successTexts`

`Interlocutor roboy.dialog.personality.states.PersonalQAState.person`

interface PhoneticEncoder

An interface for phonetic encoders that map words to their phonetic base form so that words that are written differently but sound similar receive the same form.

This is intended to be used to correct terms that Roboy misunderstood, but currently is not in use.

Subclassed by *roboy.linguistics.phonetics.DoubleMetaphoneEncoder*, *roboy.linguistics.phonetics.MetaphoneEncoder*, *roboy.linguistics.phonetics.SoundexEncoder*

Public Functions

`String roboy.linguistics.phonetics.PhoneticEncoder.encode (String input)`

class Phonetics

Public Functions

`List<String> roboy.linguistics.phonetics.Phonetics.similarWords (String word)`

Public Static Functions

`static void roboy.linguistics.phonetics.Phonetics.main (String[] args)`

Private Members

`Soundex roboy.linguistics.phonetics.Phonetics.soundex = new Soundex()`

`Map<String, List<String> > roboy.linguistics.phonetics.Phonetics.codecToWords`

class Preprocessor

Corrects abbreviated forms like “I’m” to complete forms like “I am” which are expected by later sentence analyses.

Inherits from *robey.linguistics.sentenceanalysis.Analyzer*

Public Functions

`Interpretation robey.linguistics.sentenceanalysis.Preprocessor.analyze(Interpretation i)`

class QuestionAnsweringState

State in which Roboy is answering questions based on DBpedia or the knowledge base from the resources folder.

Inherits from *robey.dialog.personality.states.State*

Public Functions

`robey.dialog.personality.states.QuestionAnsweringState.QuestionAnsweringState(State in)`

`void robey.dialog.personality.states.QuestionAnsweringState.setTop(State top)`

`List<Interpretation> robey.dialog.personality.states.QuestionAnsweringState.act()`

`Reaction robey.dialog.personality.states.QuestionAnsweringState.react(Interpretation i)`
Checks the sentence type and detected triples in the input for determining what is asked about.

Then checks its knowledge base to come up with an answer.

Private Functions

`Reaction robey.dialog.personality.states.QuestionAnsweringState.innerReaction(Interpre`

`List<Triple> robey.dialog.personality.states.QuestionAnsweringState.remember(String pr`

Private Members

`boolean robey.dialog.personality.states.QuestionAnsweringState.first = true`

`List<Triple> robey.dialog.personality.states.QuestionAnsweringState.memory`

`State robey.dialog.personality.states.QuestionAnsweringState.inner`

`State robey.dialog.personality.states.QuestionAnsweringState.top`

`List<Memory<Relation> > robey.dialog.personality.states.QuestionAnsweringState.memorie`

class QuestionAnsweringStateTest**Public Functions**

`void robey.dialog.personality.states.QuestionAnsweringStateTest.test()`

`void robey.dialog.personality.states.QuestionAnsweringStateTest.testNotAnswerable()`

`void robey.dialog.personality.states.QuestionAnsweringStateTest.testWhenWas()`

`void robey.dialog.personality.states.QuestionAnsweringStateTest.testWhereWas()`

```
void roboy.dialog.personality.states.QuestionAnsweringStateTest.testWhereDid()
void roboy.dialog.personality.states.QuestionAnsweringStateTest.testWhenDid()
void roboy.dialog.personality.states.QuestionAnsweringStateTest.testHowAdjective()
```

Private Static Attributes

```
final OpenNLPParser roboy.dialog.personality.states.QuestionAnsweringStateTest.parser =
final QuestionAnsweringState roboy.dialog.personality.states.QuestionAnsweringStateTest
```

class QuestionAskingState

Is asking the other person questions about things that we can store in the Protege memory.

Inherits from *roboy.dialog.personality.states.State*

Public Functions

```
roboy.dialog.personality.states.QuestionAskingState.QuestionAskingState(Map< String, L
List<Interpretation> roboy.dialog.personality.states.QuestionAskingState.act()
    Asks first about the name of the other person and if called another time randomly about another possible
    other information.
Reaction roboy.dialog.personality.states.QuestionAskingState.react(Interpretation input
```

Protected Functions

```
State roboy.dialog.personality.states.QuestionAskingState.determineNextState(Interpretation input)
```

Private Functions

```
Interpretation roboy.dialog.personality.states.QuestionAskingState.checkRoboyMind()
Interpretation roboy.dialog.personality.states.QuestionAskingState.checkDBpedia()
List<Interpretation> roboy.dialog.personality.states.QuestionAskingState.checkOwnMemory()
String roboy.dialog.personality.states.QuestionAskingState.analyzeObject(String sentence)
String roboy.dialog.personality.states.QuestionAskingState.analyzePredicate(String sentence)
```

Private Members

```
Concept roboy.dialog.personality.states.QuestionAskingState.objectOfFocus
String roboy.dialog.personality.states.QuestionAskingState.currentIntent
int roboy.dialog.personality.states.QuestionAskingState.questionsCount
Map<String, List<String> > roboy.dialog.personality.states.QuestionAskingState.questions
Random roboy.dialog.personality.states.QuestionAskingState.generator
Map<String, State> roboy.dialog.personality.states.QuestionAskingState.children
SmallTalkPersonality roboy.dialog.personality.states.QuestionAskingState.personality
```

Private Static Attributes

```
final int robey.dialog.personality.states.QuestionAskingState.TOASK = 2
final SimpleTokenizer robey.dialog.personality.states.QuestionAskingState.tokenizer = ne
final OpenNLPPPOSTagger robey.dialog.personality.states.QuestionAskingState.pos = new Op
final OpenNLPPParser robey.dialog.personality.states.QuestionAskingState.parser = new Open
final AnswerAnalyzer robey.dialog.personality.states.QuestionAskingState.answer = new Ans
```

class QuestionRandomizerState

Manages the questions that can be asked from a person.

Coupled with Neo4j information about the person to prevent duplicates.

Inherits from *robey.dialog.personality.states.State*

Public Functions

```
robey.dialog.personality.states.QuestionRandomizerState.QuestionRandomizerState (State
List<Interpretation> robey.dialog.personality.states.QuestionRandomizerState.act ()
Reaction robey.dialog.personality.states.QuestionRandomizerState.react (Interpretation
void robey.dialog.personality.states.QuestionRandomizerState.setTop (State top)
```

Package Attributes

```
boolean robey.dialog.personality.states.QuestionRandomizerState.askFollowUp = true
String robey.dialog.personality.states.QuestionRandomizerState.QAfile = "sentences/QAList.json"
Map<String, List<String> > robey.dialog.personality.states.QuestionRandomizerState.que
Map<String, List<String> > robey.dialog.personality.states.QuestionRandomizerState.suc
Map<String, List<String> > robey.dialog.personality.states.QuestionRandomizerState.fai
Map<String, List<String> > robey.dialog.personality.states.QuestionRandomizerState.fol
Map<String, List<String> > robey.dialog.personality.states.QuestionRandomizerState.ans
```

Private Functions

```
PersonalQAState robey.dialog.personality.states.QuestionRandomizerState.initializeQues
PersonalFollowUpState robey.dialog.personality.states.QuestionRandomizerState.initiali
void robey.dialog.personality.states.QuestionRandomizerState.checkForAskedQuestions ()
```

Private Members

```
PersonalQAState [] robey.dialog.personality.states.QuestionRandomizerState.questionSta
PersonalQAState robey.dialog.personality.states.QuestionRandomizerState.locationQuesti
PersonalFollowUpState [] robey.dialog.personality.states.QuestionRandomizerState.follo
```

```
HashMap<Neo4jRelationships, Boolean> roboy.dialog.personality.states.QuestionRandomize
State roboy.dialog.personality.states.QuestionRandomizerState.inner
State roboy.dialog.personality.states.QuestionRandomizerState.chosenState
Interlocutor roboy.dialog.personality.states.QuestionRandomizerState.person
JsonQAValues roboy.dialog.personality.states.QuestionRandomizerState.questionsAndAnswers
```

class Reaction

The reaction to what the other person said and did, consists of a list of interpretations, which is an abstraction of an utterance (the verbalizer later formulates the utterance), and a state into which the state machine moves.

Public Functions

```
roboy.dialog.personality.states.Reaction.Reaction(State state, List< Interpretation
roboy.dialog.personality.states.Reaction.Reaction(State state)
List<Interpretation> roboy.dialog.personality.states.Reaction.getReactions()
State roboy.dialog.personality.states.Reaction.getState()
void roboy.dialog.personality.states.Reaction.setState(State state)
```

Private Members

```
List<Interpretation> roboy.dialog.personality.states.Reaction.reactions
State roboy.dialog.personality.states.Reaction.state
```

class Relation

DBpedia relation.

Public Functions

```
roboy.util.Relation.Relation(Concept subject, String predicate, Concept object)
String roboy.util.Relation.getSubject()
String roboy.util.Relation.getObject()
```

Public Members

```
Concept roboy.util.Relation.subject
Concept roboy.util.Relation.object
String roboy.util.Relation.predicate
```

class Roboy

Encapsulates a *MemoryNodeModel* and enables dialog states to easily store and retrieve information about *Roboy*.

Public Functions

roboy.memory.nodes.Roboy.Roboy(String name)

Initializer for the *Roboy* node.

String roboy.memory.nodes.Roboy.getName()

Method to obtain the name of the *Roboy* node.

Return String name - text containing the name as in the *Memory*

ArrayList<Integer> roboy.memory.nodes.Roboy.getRelationships(Neo4jRelationships type)

Method to obtain the specific type relationships of the *Roboy* node.

Return ArrayList<Integer> ids - list containing integer IDs of the nodes related to the *Roboy* by specific relationship type as in the *Memory*

void roboy.memory.nodes.Roboy.addInformation(String relationship, String name)

Adds a new relation to the *Roboy* node, updating memory.

Package Attributes

Neo4jMemory roboy.memory.nodes.Roboy.memory

Private Functions

void roboy.memory.nodes.Roboy.InitializeRoboy(String name)

This method initializes the roboy property as a node that is in sync with memory and represents the *Roboy* itself.

If something goes wrong during querying, *Roboy* stays empty and soulless, and has to fallback

Private Members

MemoryNodeModel roboy.memory.nodes.Roboy.roboy

boolean roboy.memory.nodes.Roboy.memoryROS

class RoboyMind

Protege memory.

Inherits from *roboy.memory.Memory< Concept >*

Public Functions

boolean roboy.memory.RoboyMind.save(Concept object)

List<Concept> roboy.memory.RoboyMind.retrieve(Concept object)

boolean roboy.memory.RoboyMind.update(Concept object)

Map<String, List<Concept> > roboy.memory.RoboyMind.match(Concept object)

Public Members

```
int robey.memory.RoboyMind.object_id = 0
```

Public Static Functions

```
static RoboyMind robey.memory.RoboyMind.getInstance()
```

Private Functions

```
robey.memory.RoboyMind.RoboyMind()  
ServiceResponse robey.memory.RoboyMind.CreateInstance(String class_name, int object_id)  
boolean robey.memory.RoboyMind.AssertProperty(String object, String property, String value)  
List<Concept> robey.memory.RoboyMind.FindInstances(String property, String value)  
JsonObject robey.memory.RoboyMind.ListAttributes(String object)  
ServiceResponse robey.memory.RoboyMind.SaveObject(String class_name, String properties)  
Concept robey.memory.RoboyMind.GetObject(String properties, String values)  
ServiceResponse robey.memory.RoboyMind.ShowInstance(String class_name)
```

Private Static Attributes

```
RoboyMind robey.memory.RoboyMind.robeyMemory
```

```
class RoboyNameDetectionInput
```

Class detecting Roboy name.

Initiates native sphinx function of live speech analysis and checks the stream

Author Petr Romanov

Version 1.0

Date 21.04.2017

Inherits from *robey.io.InputDevice*

Public Functions

```
robey.io.RoboyNameDetectionInput.RoboyNameDetectionInput()  
    constructor which initialises recognition
```

```
void robey.io.RoboyNameDetectionInput.stopListening()  
    function for correct stopping recognition
```

```
Input robey.io.RoboyNameDetectionInput.listen()  
    tracks what was said
```

Return A signal that Roboy is one of the words in just said phrase

Protected Attributes

LiveSpeechRecognizer **roboy.io.RoboyNameDetectionInput.recog_copy**
 'link' to the object of Recognizer for correct stopping before deletion of the RoboyNameDetectorInput object

class Ros

Communication with ROS.

Public Static Functions

```
static edu.wpi.rail.jrosbridge.Ros roboy.ros.Ros.getInstance()
static void roboy.ros.Ros.close()
```

Private Functions

```
roboy.ros.Ros.Ros()
```

Private Static Attributes

```
edu.wpi.rail.jrosbridge.Ros roboy.ros.Ros.ros
final String roboy.ros.Ros.ROS_URL = System.getenv("ROS_IP")
```

enum RosClients

Stores the different client addresses and corresponding ROS message types.

Public Functions

```
roboy.ros.RosClients.RosClients(String address, String type)
```

Public Members

```
roboy.ros.RosClients.SPEECHSYNTHESIS = ("/robey/cognition/speech/synthesis/talk", Talk._TYPE)
roboy.ros.RosClients.GENERATIVE = ("/robey/cognition/generative_nlp/answer", GenerateAnswer._TYPE)
roboy.ros.RosClients.FACEDETECTION = ("/speech_synthesis/talk", DetectFace._TYPE)
roboy.ros.RosClients.OBJECTRECOGNITION = ("/speech_synthesis/talk", RecognizeObject._TYPE)
roboy.ros.RosClients.STT = ("/robey/cognition/speech/recognition", RecognizeSpeech._TYPE)
roboy.ros.RosClients.EMOTION = ("/robey/control/face/emotion", ShowEmotion._TYPE)
roboy.ros.RosClients.CREATEMEMORY = ("/robey/cognition/memory/create", DataQuery._TYPE)
roboy.ros.RosClients.UPDATEMEMORY = ("/robey/cognition/memory/update", DataQuery._TYPE)
roboy.ros.RosClients.GETMEMORY = ("/robey/cognition/memory/get", DataQuery._TYPE)
roboy.ros.RosClients.DELETEMEMORY = ("/robey/cognition/memory/remove", DataQuery._TYPE)
roboy.ros.RosClients.CYPHERMEMORY = ("/robey/cognition/memory/cypher", DataQuery._TYPE)
```

```
roboy.ros.RosClients.INTENT = ("/robey/cognition/detect_intent", DetectIntent._TYPE)
String robey.ros.RosClients.address
String robey.ros.RosClients.type
class RosMainNode
    Inherits from AbstractNodeMain
```

Public Functions

```
robey.ros.RosMainNode.RosMainNode()
GraphName robey.ros.RosMainNode.getDefaultNodeName()
void robey.ros.RosMainNode.onStart(final ConnectedNode connectedNode)
boolean robey.ros.RosMainNode.SynthesizeSpeech(String text)
String robey.ros.RosMainNode.RecognizeSpeech()
String robey.ros.RosMainNode.GenerateAnswer(String question)
boolean robey.ros.RosMainNode.ShowEmotion(String emotion)
String robey.ros.RosMainNode.CreateMemoryQuery(String query)
String robey.ros.RosMainNode.UpdateMemoryQuery(String query)
String robey.ros.RosMainNode.GetMemoryQuery(String query)
String robey.ros.RosMainNode.DeleteMemoryQuery(String query)
String robey.ros.RosMainNode.CypherMemoryQuery(String query)
Object robey.ros.RosMainNode.DetectIntent(String sentence)
```

Public Members

```
boolean robey.ros.RosMainNode.STARTUP_SUCCESS = true
```

Protected Attributes

```
Object robey.ros.RosMainNode.resp
```

Package Attributes

```
String robey.ros.RosMainNode.memoryFailure = "{" + "status : "FAIL", " + "message : "Memory client not initia
```

Private Functions

```
void robey.ros.RosMainNode.waitForLatchUnlock(CountDownLatch latch, String latchName)
    Helper method to block the calling thread until the latch is zeroed by some other task.
```

Parameters

- latch: Latch to wait for.

- `latchName`: Name to be used in log messages for the given latch.

Private Members

`CountDownLatch roboy.ros.RosMainNode.rosConnectionLatch`

`RosManager roboy.ros.RosMainNode.clients` = new *RosManager*()

class RosManager

Stores all the *Ros* Service Clients and manages access to them.

If SHUTDOWN_ON_ROS_FAILURE is set, throws a runtime exception if any of the clients failed to initialize.

Package Functions

`boolean roboy.ros.RosManager.initialize(ConnectedNode node)`

Initializes all ServiceClients for *Ros*.

`boolean roboy.ros.RosManager.notInitialized(RosClients c)`

Should always be called before getServiceClient, such that if a client failed to initialize, a fallback response can be created instead.

Important if SHUTDOWN_ON_ROS_FAILURE is false.

`ServiceClient roboy.ros.RosManager.getServiceClient(RosClients c)`

Returns the ServiceClient matching the *RosClients* entry.

the return might need casting before further use.

Private Members

`HashMap<RosClients, ServiceClient> roboy.ros.RosManager.clientMap`

class SegueState

Is checking for words with which it has associated anecdotes.

Tells the anecdote if a word matches, executes its inner state instead, if not.

Inherits from *roboy.dialog.personality.states.State*

Public Functions

`roboy.dialog.personality.states.SegueState.SegueState(State inner)`

`void roboy.dialog.personality.states.SegueState.setTop(State top)`

`List<Interpretation> roboy.dialog.personality.states.SegueState.act()`

`Reaction roboy.dialog.personality.states.SegueState.react(Interpretation input)`

Private Functions

`Reaction roboy.dialog.personality.states.SegueState.segway(Interpretation input, React`

Private Members

```
State roboy.dialog.personality.states.SegueState.inner
State roboy.dialog.personality.states.SegueState.top
Map<String,String> roboy.dialog.personality.states.SegueState.redditTIL
```

Private Static Attributes

```
final Map<SENTENCE_TYPE, Reaction> roboy.dialog.personality.states.SegueState.sentence
enum SEMANTIC_ROLE
```

Public Members

```
roboy.linguistics.Linguistics.SEMANTIC_ROLE.PREDICATE
roboy.linguistics.Linguistics.SEMANTIC_ROLE.AGENT
roboy.linguistics.Linguistics.SEMANTIC_ROLE.PATIENT
roboy.linguistics.Linguistics.SEMANTIC_ROLE.TIME
roboy.linguistics.Linguistics.SEMANTIC_ROLE.LOCATION
roboy.linguistics.Linguistics.SEMANTIC_ROLE.MANNER
roboy.linguistics.Linguistics.SEMANTIC_ROLE.INSTRUMENT
roboy.linguistics.Linguistics.SEMANTIC_ROLE.ORIGIN
roboy.linguistics.Linguistics.SEMANTIC_ROLE.DESTINATION
roboy.linguistics.Linguistics.SEMANTIC_ROLE.RECIPIENT
roboy.linguistics.Linguistics.SEMANTIC_ROLE.BENEFICIARY
roboy.linguistics.Linguistics.SEMANTIC_ROLE.PURPOSE
roboy.linguistics.Linguistics.SEMANTIC_ROLE.CAUSE
```

```
class SemanticParserAnalyzer
```

Semantic parser class.

Connects DM to Roboy parser and adds its result to interpretation class.

Inherits from *roboy.linguistics.sentenceanalysis.Analyzer*

Public Functions

```
roboy.linguistics.sentenceanalysis.SemanticParserAnalyzer.SemanticParserAnalyzer(int p
A constructor.
```

Creates ParserAnalyzer class and connects the parser to DM using a socket.

```
Interpretation roboy.linguistics.sentenceanalysis.SemanticParserAnalyzer.analyze(Interp
An analyzer function.
```

Sends input sentence to the parser and saves its response in output interpretation.

Return Input interpretation with semantic parser result.

Parameters

- `interpretation`: Input interpretation with currently processed sentence and results from previous analysis.

Private Members

Socket `roboy.linguistics.sentenceanalysis.SemanticParserAnalyzer.clientSocket`
Client socket for the parser.

PrintWriter `roboy.linguistics.sentenceanalysis.SemanticParserAnalyzer.out`
Output stream for the parser.

BufferedReader `roboy.linguistics.sentenceanalysis.SemanticParserAnalyzer.in`
Input stream from the parser.

boolean `roboy.linguistics.sentenceanalysis.SemanticParserAnalyzer.debug` = true
Boolean variable for debugging purpose.

enum `SENTENCE_TYPE`

Public Members

`roboy.linguistics.Linguistics.SENTENCE_TYPE.GREETING`

`roboy.linguistics.Linguistics.SENTENCE_TYPE.FAREWELL`

`roboy.linguistics.Linguistics.SENTENCE_TYPE.SEGUE`

`roboy.linguistics.Linguistics.SENTENCE_TYPE.ANECDOTE`

`roboy.linguistics.Linguistics.SENTENCE_TYPE.WHO`

`roboy.linguistics.Linguistics.SENTENCE_TYPE.HOW_IS`

`roboy.linguistics.Linguistics.SENTENCE_TYPE.HOW_DO`

`roboy.linguistics.Linguistics.SENTENCE_TYPE.WHY`

`roboy.linguistics.Linguistics.SENTENCE_TYPE.WHEN`

`roboy.linguistics.Linguistics.SENTENCE_TYPE.WHERE`

`roboy.linguistics.Linguistics.SENTENCE_TYPE.WHAT`

`roboy.linguistics.Linguistics.SENTENCE_TYPE.IS_IT`

`roboy.linguistics.Linguistics.SENTENCE_TYPE.DOES_IT`

`roboy.linguistics.Linguistics.SENTENCE_TYPE.STATEMENT`

`roboy.linguistics.Linguistics.SENTENCE_TYPE.NONE`

class `SentenceAnalyzer`

Tries to find triples with rather stupid heuristics and stores the results in the *Linguistics.TRIPLE* attribute of the interpretation.

Inherits from *roboy.linguistics.sentenceanalysis.Analyzer*

Public Functions

```
roboy.linguistics.sentenceanalysis.SentenceAnalyzer.SentenceAnalyzer()  
Interpretation roboy.linguistics.sentenceanalysis.SentenceAnalyzer.analyze(Interpretat
```

Private Functions

```
Interpretation roboy.linguistics.sentenceanalysis.SentenceAnalyzer.extractPAS(String s  
Triple roboy.linguistics.sentenceanalysis.SentenceAnalyzer.analyzeStatement(String[] t  
Triple roboy.linguistics.sentenceanalysis.SentenceAnalyzer.analyzeIsIt(String[] tokens  
Triple roboy.linguistics.sentenceanalysis.SentenceAnalyzer.analyzeDoesIt(String[] token  
Triple roboy.linguistics.sentenceanalysis.SentenceAnalyzer.analyzeWho(String[] tokens,  
Triple roboy.linguistics.sentenceanalysis.SentenceAnalyzer.analyzeWhat(String[] tokens  
Triple roboy.linguistics.sentenceanalysis.SentenceAnalyzer.analyzeHowIs(String[] token  
Triple roboy.linguistics.sentenceanalysis.SentenceAnalyzer.analyzeHowDo(String[] token
```

Private Members

```
Map<String,String> roboy.linguistics.sentenceanalysis.SentenceAnalyzer.meanings  
class ShutDownAction  
  Action used to shut down Roboy.
```

Sending a *ShutDownAction* leads the dialog manager to leave the communication loop in the DialogManager class and quit the program after uttering the given last words.

Inherits from *roboy.dialog.action.Action*

Public Functions

```
roboy.dialog.action.ShutDownAction.ShutDownAction(List< Action > lastwords)  
  Constructor.
```

Parameters

- lastwords: The last actions that Roboy should perform before shutting down

```
List<Action> roboy.dialog.action.ShutDownAction.getLastWords()
```

Private Members

```
List<Action> roboy.dialog.action.ShutDownAction.lastwords
```

```
class SimpleTokenizer
```

Tokenizes the text by splitting at whitespace and stores the resulting tokens in the *Linguistics.TOKENS* attribute of the interpretation.

Inherits from *roboy.linguistics.sentenceanalysis.Analyzer*

Public Functions

`Interpretation roboy.linguistics.sentenceanalysis.SimpleTokenizer.analyze(Interpretation input)`

Private Functions

`String [] roboy.linguistics.sentenceanalysis.SimpleTokenizer.tokenize(String sentence)`

class SmallTalkPersonality

Currently, Roboys main personality.

It tries to engage with people in a general small talk, remembers what was said and answers questions. The small talk personality is based on a state machine, where each input is interpreted in the context of the state Roboy is currently in to determine respective answers.

The current state machine looks like this:

Greeting state | V Introduction state | V Question Randomizer state | _Question Answering state | _Segue state | _Wild talk state

The Question Randomizer, Question Answering, Segue and Wild talk states are stacked. If one cannot give an appropriate reaction to the given utterance, the utterance is passed on to the next one. The Wild talk state will always answer.

If a farewell is uttered the personality re-initializes to the Greeting state.

What the states do: Greeting: Utters a greeting Introduction: Introduces himself and asks for the others name. Reacts differently depending on whether the other person is known. Question Randomizer: Asks the other one questions about himself and stores the answers in Roboy's memory. Question Answering: Answers questions if they are asked and Roboy knows the answer. Segue: Tells anecdotes from Today-I-Learns from Reddit if corresponding keywords are mentioned. Wild talk: Talks according to a neural network model trained on chat logs.

Inherits from *roboy.dialog.personality.Personality*

Public Functions

`roboy.dialog.personality.SmallTalkPersonality.SmallTalkPersonality(Verbalizer verbalizer)`

`List<Action> roboy.dialog.personality.SmallTalkPersonality.answer(Interpretation input)`

Reacts to inputs based on the corresponding state Roboy is in.

Each state returns a reaction to what was said and then proactively takes an action of its own. Both are combined to return the list of output actions.

`String roboy.dialog.personality.SmallTalkPersonality.getName()`

`void roboy.dialog.personality.SmallTalkPersonality.setName(String name)`

Private Functions

`void roboy.dialog.personality.SmallTalkPersonality.initialize()`

Private Members

```
String roboy.dialog.personality.SmallTalkPersonality.name
State roboy.dialog.personality.SmallTalkPersonality.state
Verbalizer roboy.dialog.personality.SmallTalkPersonality.verbalizer
RosMainNode roboy.dialog.personality.SmallTalkPersonality.rosMainNode
Interlocutor roboy.dialog.personality.SmallTalkPersonality.person
```

Private Static Attributes

```
final List<String> roboy.dialog.personality.SmallTalkPersonality.positive = Arrays.asList("er
```

class SoundexEncoder

A phonetic encoder using the method soundex that maps words to their phonetic base form so that words that are written differently but sound similar receive the same form.

This is intended to be used to correct terms that Roboy misunderstood, but currently is not is use.

Inherits from *roboy.linguistics.phonetics.PhoneticEncoder*

Public Functions

```
roboy.linguistics.phonetics.SoundexEncoder.SoundexEncoder(Soundex soundex)
String roboy.linguistics.phonetics.SoundexEncoder.encode(String input)
```

Private Members

```
Soundex roboy.linguistics.phonetics.SoundexEncoder.soundex
```

class SpeechAction

Action used for talking.

Inherits from *roboy.dialog.action.Action*

Public Functions

```
roboy.dialog.action.SpeechAction.SpeechAction(String text)
    Constructor.
```

Parameters

- `text`: The text Roboy will utter

```
String roboy.dialog.action.SpeechAction.getText()
```

Private Members

String `roboy.dialog.action.SpeechAction.text`

interface State

The central interface of the state machine.

A state always acts when its enters and reacts when its left. Both, the reaction of the last and the action of the next state, are combined to give the answer of Roboy.

Subclassed by *roboy.dialog.personality.states.AbstractBooleanState*, *roboy.dialog.personality.states.AnecdoteState*, *roboy.dialog.personality.states.CelebrityState*, *roboy.dialog.personality.states.FarewellState*, *roboy.dialog.personality.states.QuestionAnsweringState*, *roboy.dialog.personality.states.QuestionAskingState*, *roboy.dialog.personality.states.QuestionRandomizerState*, *roboy.dialog.personality.states.SegueState*, *roboy.dialog.personality.states.WildTalkState*

Public Functions

List<Interpretation> `roboy.dialog.personality.states.State.act()`

Reaction `roboy.dialog.personality.states.State.react(Interpretation input)`

class State

Central class of the dialog state system.

Every dialog state should extend this class. A state always acts when it is entered and reacts when its left. Both, the reaction of the last and the action of the next state, are combined to give the answer of Roboy.

A state can have any number of transitions to other states. Every transition has a name (like “next” or “errorState”). When designing a new state, only the transition names are known. At run time the transitions will point to other states. You can get the attached state by the transition name using `getTransition(transitionName)`.

A fallback can be attached to a state. In the case this state doesn’t know how to react to an utterance, it can return null from the `react()` function. The state machine will query the fallback in this case. More details on the fallback concept can be found in the description of the *StateBasedPersonality* and in comments below.

Subclassed by *roboy.newDialog.states.toyStates.ToyFarewellState*, *roboy.newDialog.states.toyStates.ToyGreetingsState*, *roboy.newDialog.states.toyStates.ToyIntroState*, *roboy.newDialog.states.toyStates.ToyRandomAnswerState*

Public Functions

roboy.newDialog.states.State.State(String stateIdentifier)

String `roboy.newDialog.states.State.getIdentifier()`

void `roboy.newDialog.states.State.setIdentifier(String stateIdentifier)`

final State `roboy.newDialog.states.State.getFallback()`

If this state can’t react to the input, the Personality state machine will ask the fallback state to react to the input.

This state still remains active.

Return fallback state

final void `roboy.newDialog.states.State.setFallback(State fallback)`

Set the fallback state.

The Personality state machine will ask the fallback state if this one has no answer.

Parameters

- `fallback`: fallback state

final void roboy.newDialog.states.State.setTransition(String name, State goToState)

Define a possible transition from this state to another.

Something like: “next” -> {GreetingState} “rudeInput” -> {EvilState} The next active state will be selected in *getNextState()* based on internal conditions.

Parameters

- `name`: name of the transition
- `goToState`: state to transit to

final State roboy.newDialog.states.State.getTransition(String name)

final HashMap<String, State> roboy.newDialog.states.State.getAllTransitions()

abstract List<Interpretation> roboy.newDialog.states.State.act()

A state always acts after the reaction.

Both, the reaction of the last and the action of the next state, are combined to give the answer of Roboy.

Return interpretations

abstract List<Interpretation> roboy.newDialog.states.State.react(Interpretation input)

Defines how to react to an input.

This is usually the answer to the incoming question or some other statement. If this state can’t react, it can return ‘null’ to trigger the fallback state for the answer.

Note: In the new architecture, *react()* does not define the next state anymore! Reaction and state transitions are now decoupled. *State* transitions are defined in *getNextState()*

Return reaction to the input OR null (will trigger the fallback state)

Parameters

- `input`: input from the person we talk to

abstract State roboy.newDialog.states.State.getNextState()

After this state has reacted, the personality state machine will ask this state to which state to go next.

If this state is not ready, it will return itself. Otherwise, depending on internal conditions, this state will select one of the states defined in transitions to be the next one.

Return next active state after this one has reacted

final boolean roboy.newDialog.states.State.allRequiredTransitionsAreInitialized()

Checks if all required transitions were initialized correctly.

Required transitions are defined in *getRequiredTransitionNames()*.

Return true if this state was initialized correctly

JsonObject roboy.newDialog.states.State.toJsonObject()

String roboy.newDialog.states.State.toString()

boolean roboy.newDialog.states.State.equals(Object obj)

Protected Functions

Set<String> roboy.newDialog.states.State.getRequiredTransitionNames()

Defines the names of all transition that HAVE to be defined for this state.

This function is used by *allRequiredTransitionsAreInitialized()* to make sure this state was initialized correctly. Default implementation requires no transitions to be defined.

Override this function in sub classes.

Return a set of transition names that have to be defined

Set<String> roboy.newDialog.states.State.newSet(String... tNames)

Utility function to create and initialize string sets in just one code line.

Return set initialized with inputs

Parameters

- tNames: names of the required transitions

Private Functions

boolean roboy.newDialog.states.State.equalsHelper_compareTransitions(State other)

check if every transition of this is present in the other and points to the same ID

Return true if all transitions of this state are present in the other state

Parameters

- other: other state to compare transitions

Private Members

String roboy.newDialog.states.State.stateIdentifier

State roboy.newDialog.states.State.fallback

HashMap<String, State> roboy.newDialog.states.State.transitions

class StateBasedPersonality

Implementation of Personality based on a *DialogStateMachine*.

In contrast to previous Personality implementations, this one is more generic as it loads the dialog from a file. Additionally, it is still possible to define the dialog structure directly from code (as it was done in previous implementations).

Instead of using nested states that will pass an utterance to each other if a state cannot give an appropriate reaction, we use a fallback concept. If a state doesn't know how to react, it simply doesn't react at all. If a fallback (with is another state) is attached to it, the personality will pass the utterance to the fallback automatically. This concept helps to decouple the states and reduce the dependencies between them.

Inherits from *roboy.newDialog.DialogStateMachine*, *roboy.dialog.personality.Personality*

Public Functions

roboy.newDialog.StateBasedPersonality.StateBasedPersonality(Verbalizer verb)

List<Action> roboy.newDialog.StateBasedPersonality.startConversation()

Always called once by the (new) DialogSystem at the beginning of every new conversation.

Return list of actions based on act() of the initial/active state

List<Action> roboy.newDialog.StateBasedPersonality.answer(Interpretation input)

The central method of a personality.

Given an interpretation of all inputs (audio, visual, ...) to Roboy, this method decides which actions to perform in response.

Return A list of actions to perform in response

Parameters

- input: The interpretation of the inputs

Private Functions

void roboy.newDialog.StateBasedPersonality.reset()

List<Action> roboy.newDialog.StateBasedPersonality.stateAct(State state)

Call the act function of the state and verbalize all interpretations into actions.

Return list of actions

Parameters

- state: state to call ACT on

List<Action> roboy.newDialog.StateBasedPersonality.stateReact(State state, Interpretation input)

Call the react function of the state.

If the state can't react, recursively ask fallbacks. Verbalize the resulting reaction interpretation into actions.

Return list of actions

Parameters

- state: state to call REact on
- input: input from the person Roboy speaks to

List<Action> roboy.newDialog.StateBasedPersonality.verbalizeInterpretations(List<Interpretation> interpretations)

Verbalizes all interpretations into actions using the verbalizer.

Return list of actions

Parameters

- interpretations: list of interpretations.

Private Members

```
final Verbalizer roboy.newDialog.StateBasedPersonality.verbalizer
```

class StateMachineExamples

This class provides examples how to load state machines from files or create them from code directly.

Public Static Functions

```
static void roboy.newDialog.examples.StateMachineExamples.main(String[] args)
```

Private Static Functions

```
static DialogStateMachine roboy.newDialog.examples.StateMachineExamples.fromCode()
```

```
static DialogStateMachine roboy.newDialog.examples.StateMachineExamples.fromFile()
```

```
static DialogStateMachine roboy.newDialog.examples.StateMachineExamples.fromString()
```

Private Static Attributes

```
final String roboy.newDialog.examples.StateMachineExamples.toyPersonality
```

class StatementBuilder

Public Static Functions

```
static String roboy.talk.StatementBuilder.random(List< String > list)
```

Returns a random element from the given list of Strings.

Return

Parameters

- list:

class StatementInterpreter

Public Static Functions

```
static boolean roboy.logic.StatementInterpreter.isFromList(String input, List< String > list)
```

Checks if the given String contains one of the Strings from the given list.

Return

Parameters

- input:
- list:

class Term

Public Functions

```
String roboy.linguistics.Term.toString()
```

Public Members

```
List<String> roboy.linguistics.Term.pos
```

```
float roboy.linguistics.Term.prob
```

```
String roboy.linguistics.Term.concept
```

class ToyDataGetter

Utility class to load toy data from the internet if necessary.

May be refactored into something bigger and more useful later.

Public Functions

```
roboy.linguistics.word2vec.examples.ToyDataGetter.ToyDataGetter(boolean verbose)
```

```
String roboy.linguistics.word2vec.examples.ToyDataGetter.getToyDataFilePath()
```

```
void roboy.linguistics.word2vec.examples.ToyDataGetter.ensureToyDataIsPresent()
```

Checks if toy data is present on the hard drive.

It will be downloaded if necessary.

Private Functions

```
void roboy.linguistics.word2vec.examples.ToyDataGetter.downloadData(String fromURL, St
```

```
boolean roboy.linguistics.word2vec.examples.ToyDataGetter.fileExists(String filePath)
```

Private Members

```
final boolean roboy.linguistics.word2vec.examples.ToyDataGetter.verbose
```

```
final String roboy.linguistics.word2vec.examples.ToyDataGetter.toyDataDirectory = "./resour
```

```
final String roboy.linguistics.word2vec.examples.ToyDataGetter.toyDataFilePath = "./resourc
```

```
final String roboy.linguistics.word2vec.examples.ToyDataGetter.toyDataInetURL = "https://raw
```

class ToyFarewellState

ToyFarewellState always acts with “Bye bye”.

The Interlocutor answer is ignored and there is no reaction. This ends the conversation.

Fallback is not required. This state has no outgoing transitions.

Inherits from *roboy.newDialog.states.State*

Public Functions

```
roboy.newDialog.states.toyStates.ToyFarewellState.ToyFarewellState(String stateIdentif
List<Interpretation> roboy.newDialog.states.toyStates.ToyFarewellState.act()
List<Interpretation> roboy.newDialog.states.toyStates.ToyFarewellState.react(Interpret
State roboy.newDialog.states.toyStates.ToyFarewellState.getNextState()
```

class ToyGreetingsState

ToyGreetingsState can be used as the initial state.

Roboy will greet the Interlocutor with “Hello”.

If the response is a greeting, the “next” transition is taken. Otherwise the fallback will be triggered and the “noHello” transition is taken.

Fallback is required. Outgoing transitions that have to be defined:

- next: following state if there was a greeting
- noHello: following state if there was NO greeting

Inherits from *roboy.newDialog.states.State*

Public Functions

```
roboy.newDialog.states.toyStates.ToyGreetingsState.ToyGreetingsState(String stateIdent
List<Interpretation> roboy.newDialog.states.toyStates.ToyGreetingsState.act()
List<Interpretation> roboy.newDialog.states.toyStates.ToyGreetingsState.react(Interpre
State roboy.newDialog.states.toyStates.ToyGreetingsState.getNextState()
```

Protected Functions

```
Set<String> roboy.newDialog.states.toyStates.ToyGreetingsState.getRequiredTransitionNa
```

Private Members

```
boolean roboy.newDialog.states.toyStates.ToyGreetingsState.inputOK = true
```

class ToyIntroState

ToyIntroState demonstrates a simple introduction.

Roboy will tell the Interlocutor his name and ask for the Interlocutor’s name. The reply is ignored.

Fallback is not required. Outgoing transitions that have to be defined:

- next: following state

Inherits from *roboy.newDialog.states.State*

Public Functions

```
roboy.newDialog.states.toyStates.ToyIntroState.ToyIntroState(String stateIdentifier)
List<Interpretation> roboy.newDialog.states.toyStates.ToyIntroState.act()
List<Interpretation> roboy.newDialog.states.toyStates.ToyIntroState.react(Interpretation)
State roboy.newDialog.states.toyStates.ToyIntroState.getNextState()
```

Protected Functions

```
Set<String> roboy.newDialog.states.toyStates.ToyIntroState.getRequiredTransitionNames()
```

class ToyRandomAnswerState

ToyRandomAnswerState is meant to be used as a fallback state.

It only implements the react() function returning a hardcoded random answer. This state should never become active (meaning that no transition should point to it.)

Fallback is not required (this state should be the fallback). This state has no outgoing transitions.

Inherits from *roboy.newDialog.states.State*

Public Functions

```
roboy.newDialog.states.toyStates.ToyRandomAnswerState.ToyRandomAnswerState(String stateIdentifier)
List<Interpretation> roboy.newDialog.states.toyStates.ToyRandomAnswerState.act()
List<Interpretation> roboy.newDialog.states.toyStates.ToyRandomAnswerState.react(Interpretation)
State roboy.newDialog.states.toyStates.ToyRandomAnswerState.getNextState()
```

class ToyStateFactory

Temporary factory to create *State* objects based on class name.

May be replaced with something more generic later.

Public Static Functions

```
static State roboy.newDialog.states.factories.ToyStateFactory.getByClassName(String className)
```

class Triple

Represents a simple who(agens) does what(predicate) to whom(patients) relation.

Public Functions

```
roboy.linguistics.Triple.Triple(String predicate, String agens, String patients)
String roboy.linguistics.Triple.toString()
```

Public Members

```
String robby.linguistics.Triple.agens
String robby.linguistics.Triple.predicate
String robby.linguistics.Triple.patients
```

class UdpInput

Created by robby on 7/27/17.

Inherits from *robby.io.InputDevice*

Public Functions

```
robby.io.UdpInput.UdpInput (DatagramSocket ds)
Input robby.io.UdpInput.listen()
```

Private Members

```
DatagramSocket robby.io.UdpInput.serverSocket
```

class UdpOutput

Created by robby on 7/27/17.

Inherits from *robby.io.OutputDevice*

Public Functions

```
robby.io.UdpOutput.UdpOutput (DatagramSocket ds, String address, int port)
void robby.io.UdpOutput.act (List< Action > actions)
```

Private Members

```
DatagramSocket robby.io.UdpOutput.serverSocket
InetAddress robby.io.UdpOutput.udpEndpointAddress
int robby.io.UdpOutput.udpEndpointPort
```

enum Updaters

All available updaters by their class and their target's value type.

Public Functions

```
robby.context.Context.Updaters.Updaters (Class attribute, Class valueType)
```

Public Members

```
roboy.context.Context.Updates.DIALOG_TOPICS_UPDATER=(DialogTopicsUpdater.class, String.class)
final Class roboy.context.Context.Updates.classType
final Class roboy.context.Context.Updates.targetValueType
```

class Util

Helper class.

Inherits from Exception

Public Static Functions

```
static String roboy.memory.Util.getPartURI (String URI)
static List<String> roboy.memory.Util.getQuestionType (String question)
static int roboy.memory.Util.calculateLevenshteinDistance (String s, String t)
static int roboy.memory.Util.min (int a, int b, int c)
```

template <V>

class Value

Stores a single value of type V.

Inherits from *roboy::context::AbstractValue< V >*

Subclassed by *roboy.context.contextObjects.FaceCoordinates*

Public Functions

```
V roboy.context.Value< V >.getValue ()
void roboy.context.Value< V >.updateValue (V value)
```

Private Members

```
volatile V roboy.context.Value< V >.value = null
```

enum ValueHistories

All available valueHistories.

Inherits from *roboy.context.ExternalContextInterface*

Public Functions

```
roboy.context.Context.ValueHistories.ValueHistories (Class<?extends AbstractValueHisto
Class roboy.context.Context.ValueHistories.getClassType ()
Class roboy.context.Context.ValueHistories.getReturnType ()
public<T> T roboy.context.Context.ValueHistories.getLastValue ()
public<K, T> Map<K, T> roboy.context.Context.ValueHistories.getNLastValues (int n)
```

Public Members

```
roboy.context.Context.ValueHistories.DIALOG_TOPICS=(DialogTopics.class, String.class)
final Class roboy.context.Context.ValueHistories.classType
final Class roboy.context.Context.ValueHistories.returnType
template <V>
class ValueHistory
    HashMap implementation of a value history with unique Integer keys.
    Inherits from roboy.context.AbstractValueHistory< Integer, V >
    Subclassed by roboy.context.contextObjects.DialogTopics
```

Public Functions

```
roboy.context.ValueHistory< V >.ValueHistory()
V roboy.context.ValueHistory< V >.getValue() Return
    The last element added to this list.

HashMap<Integer, V> roboy.context.ValueHistory< V >.getLastNValues(int n)
    Get a copy of the last n entries added to the list.

    Less values may be returned if there are not enough values in this list. In case of no values, an empty array
    is returned.

    Return A hashmap of n last values added to the list.

    Parameters
        • n: The number of instances to retrieve.

synchronized void roboy.context.ValueHistory< V >.updateValue(V value)
    Puts a value into the list in the last place.

    Parameters
        • value: The value to be added.
```

Private Functions

```
synchronized int roboy.context.ValueHistory< V >.generateKey()
    Generates a key that is unique through incrementing an internal counter.

    Return A key which is unique in this list instance.

synchronized V roboy.context.ValueHistory< V >.getValue(int key)
    In a ValueList, only getValue() and updateValue() directly access the HashMap data.

    Setting these methods to be synchronous avoids concurrency issues.

    Return The value, or null if not found.

    Parameters
        • key: The key of the value.
```

Private Members

volatile int roboy.context.ValueHistory< V >.counter

This counter tracks the number of values, indices still start from 0.

Reading is allowed without synchronization, modifications only through *generateKey()*.

HashMap<Integer, V> roboy.context.ValueHistory< V >.data

enum Values

All available values.

Inherits from *roboy.context.ExternalContextInterface*

Public Functions

roboy.context.Context.Values.Values(Class<?extends AbstractValue > attribute, Class <

Class roboy.context.Context.Values.getClassType()

Class roboy.context.Context.Values.getReturnType()

public<T> T roboy.context.Context.Values.getLastValue()

Public Members

roboy.context.Context.Values.FACE_COORDINATES=(FaceCoordinates.class, CoordinateSet.class)

final Class roboy.context.Context.Values.classType

final Class roboy.context.Context.Values.returnType

class Verbalizer

Turns interpretations to actual utterances.

This should in the future lead to diversifying the ways Roboy is expressing information.

Public Functions

Action roboy.talk.Verbalizer.verbalize(Interpretation interpretation)

Currently contains utterance diversification for greetings, farewells, segue and introductions to anecdotes.

In all other cases the state machine provides a literal sentence that is just passed through. In the future, this should be extended to diversify everything Roboy says.

Return the actual action that is performed

Parameters

- *interpretation*: the abstraction of what Roboy intends to say

Public Static Attributes

final List<String> roboy.talk.Verbalizer.greetings = Arrays.asList("hello","hi","greetings","good mornin

final List<String> roboy.talk.Verbalizer.farewells = Arrays.asList("ciao","goodbye","cheerio","bye","se

Private Functions

```

SpeechAction roboy.talk.Verbalizer.greet(Interpretation interpretation)
ShutdownAction roboy.talk.Verbalizer.farewell(Interpretation interpretation)
SpeechAction roboy.talk.Verbalizer.segue(Interpretation interpretation)
SpeechAction roboy.talk.Verbalizer.anecdote(Interpretation interpretation)
Interpretation roboy.talk.Verbalizer.verbalizeDates(Interpretation interpretation)
String roboy.talk.Verbalizer.dateToText(String date)
SpeechAction roboy.talk.Verbalizer.literalSentence(Interpretation interpretation)

```

Private Static Attributes

```

final List<String> roboy.talk.Verbalizer.segues = Arrays.asList("talking about ","since you mentioned ","")
final List<String> roboy.talk.Verbalizer.preAnecdotes = Arrays.asList("here is an interesting bit of trivia")
final List<String> roboy.talk.Verbalizer.anecdotes = Arrays.asList("did you know ","did you know that ")
final Map<String, String> roboy.talk.Verbalizer.dayNumberMap
final Map<Integer, String> roboy.talk.Verbalizer.lowNumberMap = Maps.intStringMap( 1,"one", 2,"two")
final Map<String, String> roboy.talk.Verbalizer.monthNumberMap = Maps.stringMap( "1","January", "2","February")
final Map<Integer, String> roboy.talk.Verbalizer.tenthNumberMap = Maps.intStringMap( 1,"ten", 2,"twenty")
class VerbalizerTest

```

Public Functions

```

void roboy.talk.VerbalizerTest.testDates()
class Vision
  Vision helper class.

```

Public Functions

```

String roboy.io.Vision.recognizeFace()
boolean roboy.io.Vision.findFaces()

```

Public Static Functions

```

static Vision roboy.io.Vision.getInstance()

```

Private Functions

```

roboy.io.Vision.Vision()

```

Private Static Attributes

`Vision roboy.io.Vision.roboyVision`

class VisionCallback

Inherits from TopicCallback

Public Functions

`void roboy.io.Vision.VisionCallback.handleMessage(Message message)`

Public Members

`String roboy.io.Vision.VisionCallback.latest = null`

`boolean roboy.io.Vision.VisionCallback.faceDetected = false`

class WildTalkState

The generative model talking wildly.

Inherits from *roboy.dialog.personality.states.State*

Public Functions

`roboy.dialog.personality.states.WildTalkState.WildTalkState(RosMainNode node)`

`List<Interpretation> roboy.dialog.personality.states.WildTalkState.act()`

`Reaction roboy.dialog.personality.states.WildTalkState.react(Interpretation input)`

`void roboy.dialog.personality.states.WildTalkState.setNextState(State next)`

Private Members

`State roboy.dialog.personality.states.WildTalkState.next = this`

`RosMainNode roboy.dialog.personality.states.WildTalkState.rosMainNode`

class Word2vecTrainingExample

Neural net that processes text into word-vectors.

Adapted from `org.deeplearning4j.examples.nlp.word2vec.Word2VecRawTextExample`

Public Static Functions

`static void roboy.linguistics.word2vec.examples.Word2vecTrainingExample.main(String[] args)`

class Word2vecUptrainingExample

Neural net that processes text into word-vectors.

This example shows how to save/load and train the model.

Adapted from `org.deeplearning4j.examples.nlp.word2vec.Word2VecUptrainingExample`

Public Static Functions

```
static void robey.linguistics.word2vec.examples.Word2vecUptrainingExample.main(String[] args)
class WorkingMemory
  Inherits from robey.memory.Memory< Triple >
```

Public Functions

```
boolean robey.memory.WorkingMemory.save(Triple object)
String robey.memory.WorkingMemory.toString()
List<Triple> robey.memory.WorkingMemory.retrieve(Triple object)
```

Public Static Functions

```
static WorkingMemory robey.memory.WorkingMemory.getInstance()
```

Private Functions

```
robey.memory.WorkingMemory.WorkingMemory()
void robey.memory.WorkingMemory.addToMap(Map< String, List< Triple >> list, String s)
```

Private Members

```
Map<String,List<Triple> > robey.memory.WorkingMemory.agensTripleMap = new HashMap<>()
Map<String,List<Triple> > robey.memory.WorkingMemory.patientsTripleMap = new HashMap<>()
Map<String,List<Triple> > robey.memory.WorkingMemory.predicateTripleMap = new HashMap<>()
```

Private Static Attributes

```
WorkingMemory robey.memory.WorkingMemory.memory
namespace gson
namespace api
namespace awt
namespace io
namespace net
namespace util
namespace swing
namespace query
namespace model
namespace sparql
```

namespace Assert
namespace node
namespace roboy
namespace context
namespace contextObjects
namespace GUI
namespace visionContext
namespace dialog
namespace action
namespace ConfigurationProfile
namespace personality
namespace states
namespace io
namespace linguistics
namespace phonetics
namespace sentenceanalysis
namespace word2vec
namespace examples
namespace logic
namespace memory
namespace Neo4jRelationships
namespace nodes
namespace newDialog
namespace examples
namespace states
namespace factories
namespace toyStates
namespace ros
namespace talk
namespace util
namespace roboy_communication_cognition
namespace roboy_communication_control
file AbstractValue.java
file AbstractValueHistory.java
file AttributeManager.java
file Context.java

file `CoordinateSet.java`
file `DialogTopics.java`
file `DialogTopicsUpdater.java`
file `FaceCoordinates.java`
file `FaceCoordinatesUpdater.java`
file `ExternalContextInterface.java`
file `ExternalUpdater.java`
file `ContextGUI.java`
file `InternalUpdater.java`
file `IntervalUpdater.java`
file `Value.java`
file `ValueHistory.java`
file `Action.java`
file `FaceAction.java`
file `ShutDownAction.java`
file `SpeechAction.java`
file `Config.java`
file `DialogSystem.java`
file `CuriousPersonality.java`
file `DefaultPersonality.java`
file `KnockKnockPersonality.java`
file `Personality.java`
file `SmallTalkPersonality.java`
file `AbstractBooleanState.java`
file `AnecdoteState.java`
file `CelebrityState.java`
file `ConverseState.java`
file `FarewellState.java`
file `GenerativeCommunicationState.java`
file `GreetingState.java`
file `IdleState.java`
file `InquiryState.java`
file `IntroductionState.java`
file `LocationDBpedia.java`
file `PersonalFollowUpState.java`
file `PersonalQAState.java`

file **QuestionAnsweringState.java**
file **QuestionAskingState.java**
file **QuestionRandomizerState.java**
file **Reaction.java**
file **SegueState.java**
file **State.java**
file **State.java**
file **WildTalkState.java**
file **BingInput.java**
file **BingOutput.java**
file **CelebritySimilarityInput.java**
file **CerevoiceOutput.java**
file **CommandLineCommunication.java**
file **CommandLineInput.java**
file **CommandLineOutput.java**
file **Communication.java**
file **EmotionOutput.java**
file **FreeTTSOutput.java**
file **Input.java**
file **InputDevice.java**
file **MultiInputDevice.java**
file **MultiOutputDevice.java**
file **OutputDevice.java**
file **RoboyNameDetectionInput.java**
file **UdpInput.java**
file **UdpOutput.java**
file **Vision.java**
file **Concept.java**
file **Concept.java**
file **DetectedEntity.java**
file **Entity.java**
file **Linguistics.java**
file **DoubleMetaphoneEncoder.java**
file **MetaphoneEncoder.java**
file **PhoneticEncoder.java**
file **Phonetics.java**

file SoundexEncoder.java
file Analyzer.java
file AnswerAnalyzer.java
file DictionaryBasedSentenceTypeDetector.java
file EmotionAnalyzer.java
file IntentAnalyzer.java
file Interpretation.java
file OntologyNERAnalyzer.java
file OpenNLPParser.java
file OpenNLPPPOSTagger.java
file Preprocessor.java
file SemanticParserAnalyzer.java
file SentenceAnalyzer.java
file SimpleTokenizer.java
file Term.java
file Triple.java
file ToyDataGetter.java
file Word2vecTrainingExample.java
file Word2vecUptrainingExample.java
file Intention.java
file IntentionClassifier.java
file PASInterpreter.java
file StatementInterpreter.java
file DBpediaMemory.java
file Lexicon.java
file LexiconLiteral.java
file LexiconPredicate.java
file Memory.java
file Neo4jMemory.java
file Neo4jRelationships.java
file Interlocutor.java
file MemoryNodeModel.java
file Roboy.java
file PersistentKnowledge.java
file RoboyMind.java
file Util.java

file **WorkingMemory.java**

file **DialogStateMachine.java**

file **StateMachineExamples.java**

file **NewDialogSystem.java**

file **StateBasedPersonality.java**

file **ToyStateFactory.java**

file **ToyFarewellState.java**

file **ToyGreetingsState.java**

file **ToyIntroState.java**

file **ToyRandomAnswerState.java**

file **Ros.java**

file **RosClients.java**

file **RosMainNode.java**

file **RosManager.java**

file **StatementBuilder.java**

file **Verbalizer.java**

file **IO.java**

file **JsonQAValues.java**

file **JsonUtils.java**

file **Lists.java**

file **Maps.java**

file **Relation.java**

file **ContextTest.java**

file **LocationDBpediaStateTest.java**

file **QuestionAnsweringStateTest.java**

file **AnswerAnalyzerTest.java**

file **DictionaryBasedSentenceTypeDetectorTest.java**

file **OpenNLPParserTest.java**

file **PASInterpreterTest.java**

file **DialogStateMachineTest.java**

file **VerbalizerTest.java**

dir **/home/docs/checkouts/readthedocs.org/user_builds/robeydialog/checkouts/stable/src/main/j**

dir **/home/docs/checkouts/readthedocs.org/user_builds/robeydialog/checkouts/stable/src/test/j**

dir **/home/docs/checkouts/readthedocs.org/user_builds/robeydialog/checkouts/stable/src/main/j**

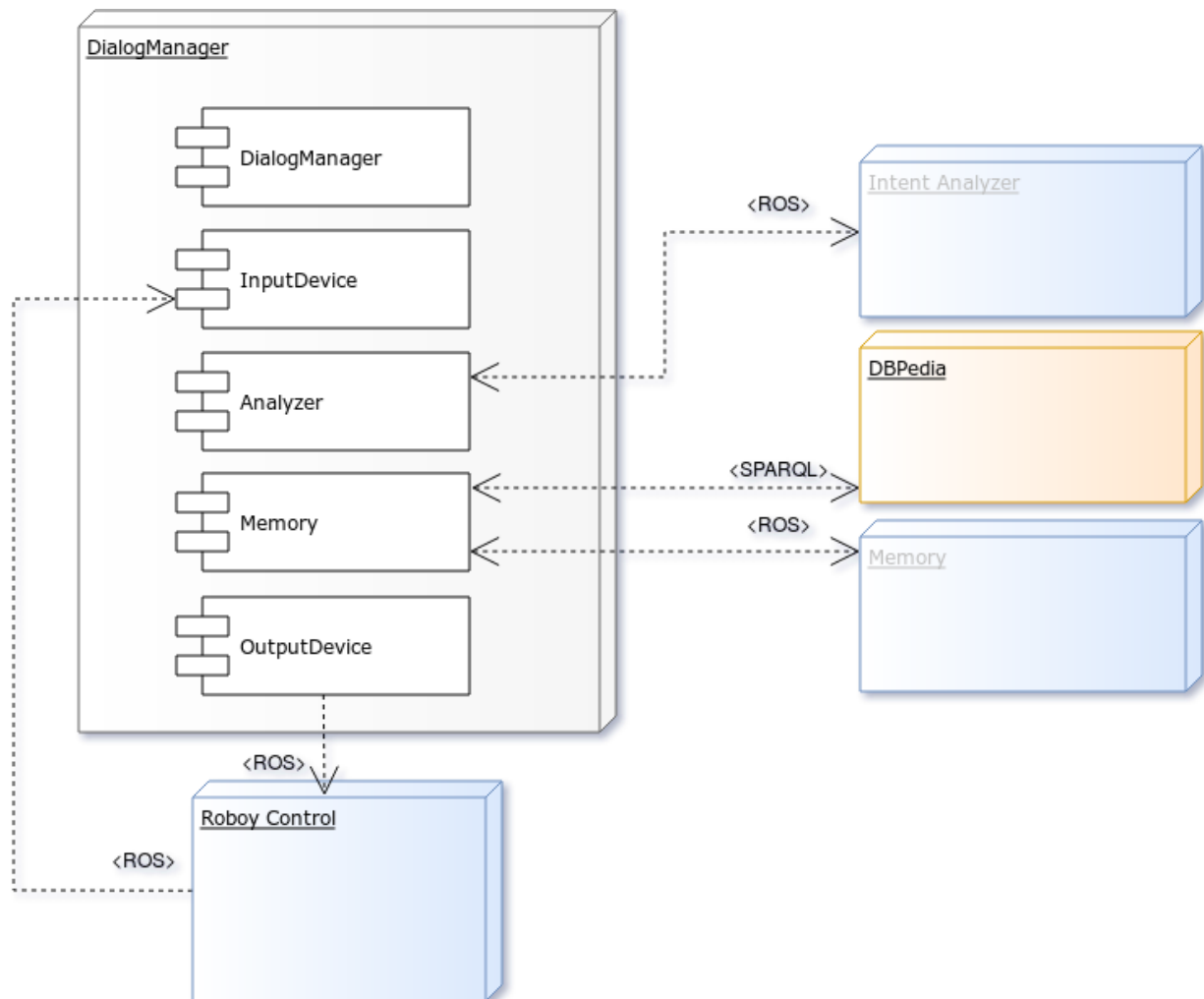
dir **/home/docs/checkouts/readthedocs.org/user_builds/robeydialog/checkouts/stable/src/main/j**

dir **/home/docs/checkouts/readthedocs.org/user_builds/robeydialog/checkouts/stable/src/test/j**

[illegible]

2.9 Deployment diagram

This diagram depicts the external modules and their communication channels/protocols with the Dialog Manager. Modules which are in development (intents, Memory) are included in gray. For simplicity, the Dialog Manager module only contains these components which are most relevant for external connections.



2.10 Sequence diagram

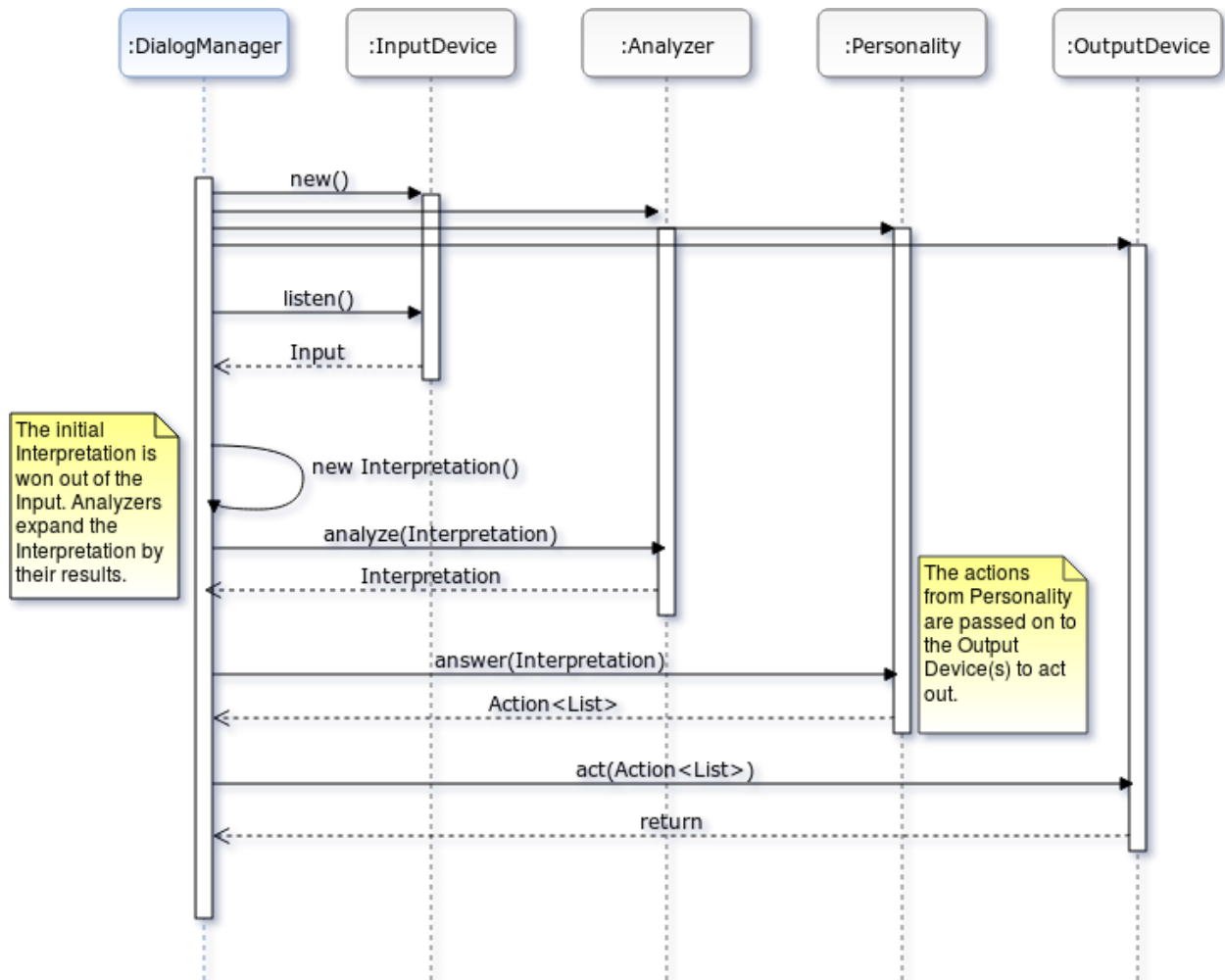
This is a simple high-abstraction sequence of the Dialog Manager's workflow, starting at the initialization and going from registering input over interpretation and action generation to sending output actions.

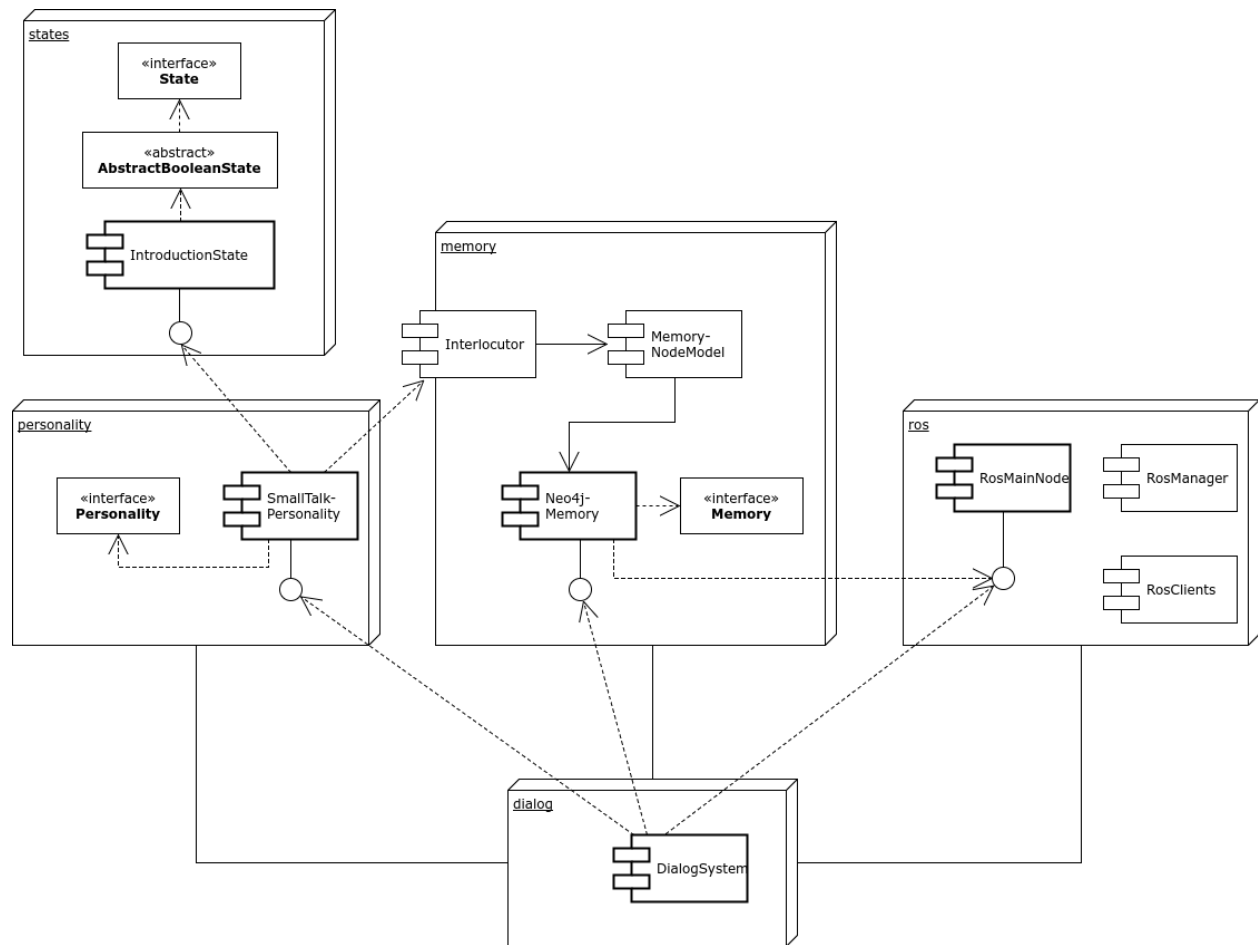
alt Sequence diagram

2.11 Building block diagram

This diagram depicts the internal modules of the Dialog System and their dependency hierarchies.

alt Building block diagram





2.12 About arc42

This information should stay in every repository as per their license: <http://www.arc42.de/template/licence.html>

arc42, the Template for documentation of software and system architecture.

By Dr. Gernot Starke, Dr. Peter Hruschka and contributors.

Template Revision: 6.5 EN (based on asciidoc), Juni 2014

© We acknowledge that this document uses material from the arc 42 architecture template, <http://www.arc42.de>. Created by Dr. Peter Hruschka & Dr. Gernot Starke. For additional contributors see <http://arc42.de/sonstiges/contributors.html>

Note

This version of the template contains some help and explanations. It is used for familiarization with arc42 and the understanding of the concepts. For documentation of your own system you use better the *plain* version.

2.12.1 Literature and references

Starke-2014 Gernot Starke: Effektive Softwarearchitekturen - Ein praktischer Leitfaden. Carl Hanser Verlag, 6, Auflage 2014.

Starke-Hruschka-2011 Gernot Starke und Peter Hruschka: Softwarearchitektur kompakt. Springer Akademischer Verlag, 2. Auflage 2011.

Zörner-2013 Softwarearchitekturen dokumentieren und kommunizieren, Carl Hanser Verlag, 2012

2.12.2 Examples

- [HTML Sanity Checker](#)
- [DocChess](#) (german)
- [Gradle](#) (german)
- [MaMa CRM](#) (german)
- [Financial Data Migration](#) (german)

2.12.3 Acknowledgements and collaborations

arc42 originally envisioned by [Dr. Peter Hruschka](#) and [Dr. Gernot Starke](#).

Sources We maintain arc42 in *asciidoc* format at the moment, hosted in [GitHub](#) under the [aim42-Organisation](#).

Issues We maintain a list of [open topics](#) and [bugs](#).

We are looking forward to your corrections and clarifications! Please fork the repository mentioned over this lines and send us a *pull request*!

2.12.4 Collaborators

We are very thankful and acknowledge the support and help provided by all active and former collaborators, uncountable (anonymous) advisors, bug finders and users of this method.

Currently active

- Gernot Starke
- Stefan Zörner
- Markus Schärtel
- Ralf D. Müller
- Peter Hruschka
- Jürgen Krey

Former collaborators

(in alphabetical order)

- Anne Aloysius
- Matthias Bohlen
- Karl Eilebrecht
- Manfred Ferken
- Phillip Ghadir
- Carsten Klein
- Prof. Arne Koschel
- Axel Scheithauer

C

com::google::gson (C++ type), 85

E

edu::cmu::sphinx::api (C++ type), 85

J

java::awt (C++ type), 85

java::io (C++ type), 85

java::net (C++ type), 85

java::util (C++ type), 85

javax::swing (C++ type), 85

O

org::apache::jena::query (C++ type), 85

org::apache::jena::rdf::model (C++ type), 85

org::apache::jena::sparql (C++ type), 85

org::junit::Assert (C++ type), 85

org::ros::node (C++ type), 86

R

roboy (C++ type), 86

roboy::context (C++ type), 86

roboy::context::AbstractValue (C++ class), 15

roboy::context::AbstractValueHistory (C++ class), 15

roboy::context::AttributeManager (C++ class), 17

roboy::context::Context (C++ class), 23

roboy::context::Context::Updaters (C++ type), 79

roboy::context::Context::ValueHistories (C++ type), 80

roboy::context::Context::Values (C++ type), 82

roboy::context::contextObjects (C++ type), 86

roboy::context::contextObjects::CoordinateSet (C++ class), 25

roboy::context::contextObjects::DialogTopics (C++ class), 31

roboy::context::contextObjects::DialogTopicsUpdater (C++ class), 31

roboy::context::contextObjects::FaceCoordinates (C++ class), 34

roboy::context::contextObjects::FaceCoordinatesUpdater (C++ class), 34

roboy::context::ExternalContextInterface (C++ class), 32

roboy::context::ExternalUpdater (C++ class), 33

roboy::context::GUI (C++ type), 86

roboy::context::GUI::ContextGUI (C++ class), 24

roboy::context::InternalUpdater (C++ class), 38

roboy::context::IntervalUpdater (C++ class), 39

roboy::context::Value (C++ class), 80

roboy::context::ValueHistory (C++ class), 81

roboy::context::visionContext (C++ type), 86

roboy::context::visionContext::ContextTest (C++ class), 24

roboy::dialog (C++ type), 86

roboy::dialog::action (C++ type), 86

roboy::dialog::action::Action (C++ class), 16

roboy::dialog::action::FaceAction (C++ class), 33

roboy::dialog::action::ShutDownAction (C++ class), 68

roboy::dialog::action::SpeechAction (C++ class), 70

roboy::dialog::Config (C++ class), 21

roboy::dialog::Config::ConfigurationProfile (C++ type), 22, 86

roboy::dialog::DialogSystem (C++ class), 29

roboy::dialog::personality (C++ type), 86

roboy::dialog::personality::CuriousPersonality (C++ class), 25

roboy::dialog::personality::DefaultPersonality (C++ class), 27

roboy::dialog::personality::DefaultPersonality::CONVERSATIONAL_STATE (C++ type), 25

roboy::dialog::personality::KnockKnockPersonality (C++ class), 42

roboy::dialog::personality::KnockKnockPersonality::KnockKnockState (C++ type), 42

roboy::dialog::personality::Personality (C++ class), 55

roboy::dialog::personality::SmallTalkPersonality (C++ class), 69

roboy::dialog::personality::states (C++ type), 86

roboy::dialog::personality::states::AbstractBooleanState (C++ class), 14

robey::dialog::personality::states::AnecdoteState (C++ class), 16	robey::io::MultiInputDevice (C++ class), 48
robey::dialog::personality::states::CelebrityState (C++ class), 18	robey::io::MultiOutputDevice (C++ class), 48
robey::dialog::personality::states::ConverseState (C++ class), 25	robey::io::OutputDevice (C++ class), 53
robey::dialog::personality::states::FarewellState (C++ class), 34	robey::io::RoboyNameDetectionInput (C++ class), 62
robey::dialog::personality::states::GenerativeCommunicationState (C++ class), 35	robey::io::UdpInput (C++ class), 79
robey::dialog::personality::states::GreetingState (C++ class), 35	robey::io::UdpOutput (C++ class), 79
robey::dialog::personality::states::IdleState (C++ class), 35	robey::io::Vision (C++ class), 83
robey::dialog::personality::states::InquiryState (C++ class), 36	robey::io::Vision::VisionCallback (C++ class), 84
robey::dialog::personality::states::IntroductionState (C++ class), 40	robey::linguistics (C++ type), 86
robey::dialog::personality::states::LocationDBpedia (C++ class), 46	robey::linguistics::Concept (C++ class), 21
robey::dialog::personality::states::LocationDBpediaStateTest (C++ class), 46	robey::linguistics::DetectedEntity (C++ class), 28
robey::dialog::personality::states::PersonalFollowUpState (C++ class), 54	robey::linguistics::Entity (C++ class), 32
robey::dialog::personality::states::PersonalQAState (C++ class), 55	robey::linguistics::Linguistics (C++ class), 44
robey::dialog::personality::states::QuestionAnsweringState (C++ class), 57	robey::linguistics::Linguistics::SEMANTIC_ROLE (C++ type), 66
robey::dialog::personality::states::QuestionAnsweringStateTest (C++ class), 57	robey::linguistics::Linguistics::SENTENCE_TYPE (C++ type), 67
robey::dialog::personality::states::QuestionAskingState (C++ class), 58	robey::linguistics::phonetics (C++ type), 86
robey::dialog::personality::states::QuestionRandomizerState (C++ class), 59	robey::linguistics::phonetics::DoubleMetaphoneEncoder (C++ class), 31
robey::dialog::personality::states::Reaction (C++ class), 60	robey::linguistics::phonetics::MetaphoneEncoder (C++ class), 48
robey::dialog::personality::states::SegueState (C++ class), 65	robey::linguistics::phonetics::PhoneticEncoder (C++ class), 56
robey::dialog::personality::states::State (C++ class), 71	robey::linguistics::phonetics::Phonetics (C++ class), 56
robey::dialog::personality::states::WildTalkState (C++ class), 84	robey::linguistics::phonetics::SoundexEncoder (C++ class), 70
robey::io (C++ type), 86	robey::linguistics::sentenceanalysis (C++ type), 86
robey::io::BingInput (C++ class), 18	robey::linguistics::sentenceanalysis::Analyzer (C++ class), 16
robey::io::BingOutput (C++ class), 18	robey::linguistics::sentenceanalysis::AnswerAnalyzer (C++ class), 16
robey::io::CelebritySimilarityInput (C++ class), 18	robey::linguistics::sentenceanalysis::AnswerAnalyzerTest (C++ class), 17
robey::io::CerevoiceOutput (C++ class), 19	robey::linguistics::sentenceanalysis::DictionaryBasedSentenceTypeDetector (C++ class), 31
robey::io::CommandLineCommunication (C++ class), 19	robey::linguistics::sentenceanalysis::DictionaryBasedSentenceTypeDetector (C++ class), 31
robey::io::CommandLineInput (C++ class), 20	robey::linguistics::sentenceanalysis::EmotionAnalyzer (C++ class), 32
robey::io::CommandLineOutput (C++ class), 20	robey::linguistics::sentenceanalysis::IntentAnalyzer (C++ class), 36
robey::io::Communication (C++ class), 20	robey::linguistics::sentenceanalysis::Interpretation (C++ class), 38
robey::io::EmotionOutput (C++ class), 32	robey::linguistics::sentenceanalysis::OntologyNERAnalyzer (C++ class), 51
robey::io::FreeTTSOutput (C++ class), 34	robey::linguistics::sentenceanalysis::OpenNLPParser (C++ class), 51
robey::io::Input (C++ class), 35	robey::linguistics::sentenceanalysis::OpenNLPParserTest (C++ class), 52
robey::io::InputDevice (C++ class), 36	robey::linguistics::sentenceanalysis::OpenNLPPPOSTagger (C++ class), 52
	robey::linguistics::sentenceanalysis::Preprocessor (C++ class), 56

robpy::linguistics::sentenceanalysis::SemanticParserAnalyzer (C++ class), 66
 robpy::linguistics::sentenceanalysis::SentenceAnalyzer (C++ class), 67
 robpy::linguistics::sentenceanalysis::SimpleTokenizer (C++ class), 68
 robpy::linguistics::Term (C++ class), 75
 robpy::linguistics::Triple (C++ class), 78
 robpy::linguistics::word2vec (C++ type), 86
 robpy::linguistics::word2vec::examples (C++ type), 86
 robpy::linguistics::word2vec::examples::ToyDataGetter (C++ class), 76
 robpy::linguistics::word2vec::examples::Word2vecTrainingExample (C++ class), 84
 robpy::linguistics::word2vec::examples::Word2vecUptrainingExample (C++ class), 84
 robpy::logic (C++ type), 86
 robpy::logic::Intention (C++ class), 37
 robpy::logic::IntentionClassifier (C++ class), 37
 robpy::logic::PASInterpreter (C++ class), 53
 robpy::logic::PASInterpreterTest (C++ class), 53
 robpy::logic::StatementInterpreter (C++ class), 75
 robpy::memory (C++ type), 86
 robpy::memory::DBpediaMemory (C++ class), 26
 robpy::memory::Lexicon (C++ class), 42
 robpy::memory::LexiconLiteral (C++ class), 43
 robpy::memory::LexiconPredicate (C++ class), 44
 robpy::memory::Memory (C++ class), 46
 robpy::memory::Neo4jMemory (C++ class), 49
 robpy::memory::Neo4jRelationships (C++ type), 50, 86
 robpy::memory::nodes (C++ type), 86
 robpy::memory::nodes::Interlocutor (C++ class), 37
 robpy::memory::nodes::MemoryNodeModel (C++ class), 47
 robpy::memory::nodes::Roboy (C++ class), 60
 robpy::memory::PersistentKnowledge (C++ class), 54
 robpy::memory::RoboyMind (C++ class), 61
 robpy::memory::Util (C++ class), 80
 robpy::memory::WorkingMemory (C++ class), 85
 robpy::newDialog (C++ type), 86
 robpy::newDialog::DialogStateMachine (C++ class), 28
 robpy::newDialog::DialogStateMachineTest (C++ class), 29
 robpy::newDialog::examples (C++ type), 86
 robpy::newDialog::examples::StateMachineExamples (C++ class), 75
 robpy::newDialog::NewDialogSystem (C++ class), 51
 robpy::newDialog::StateBasedPersonality (C++ class), 73
 robpy::newDialog::states (C++ type), 86
 robpy::newDialog::states::factories (C++ type), 86
 robpy::newDialog::states::factories::ToyStateFactory (C++ class), 78
 robpy::newDialog::states::State (C++ class), 71
 robpy::newDialog::states::toyStates (C++ type), 86
 robpy::newDialog::states::toyStates::ToyFarewellState (C++ class), 76
 robpy::newDialog::states::toyStates::ToyGreetingsState (C++ class), 77
 robpy::newDialog::states::toyStates::ToyIntroState (C++ class), 77
 robpy::newDialog::states::toyStates::ToyRandomAnswerState (C++ class), 78
 robpy::ros (C++ type), 86
 robpy::ros::Ros (C++ class), 63
 robpy::ros::RosClients (C++ type), 63
 robpy::ros::RosMainNode (C++ class), 64
 robpy::ros::RosManager (C++ class), 65
 robpy::talk (C++ type), 86
 robpy::talk::StatementBuilder (C++ class), 75
 robpy::talk::Verbalizer (C++ class), 82
 robpy::talk::VerbalizerTest (C++ class), 83
 robpy::util (C++ type), 86
 robpy::util::Concept (C++ class), 20
 robpy::util::IO (C++ class), 40
 robpy::util::JsonEntryModel (C++ class), 40
 robpy::util::JsonModel (C++ class), 41
 robpy::util::JsonQAVvalues (C++ class), 41
 robpy::util::JsonUtils (C++ class), 41
 robpy::util::Lists (C++ class), 45
 robpy::util::Maps (C++ class), 46
 robpy::util::Relation (C++ class), 60
 robpy_communication_cognition (C++ type), 86
 robpy_communication_control (C++ type), 86