
Roboy Dialog System

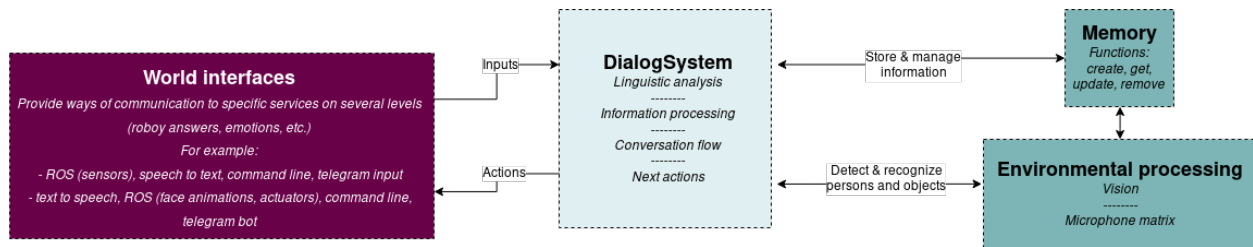
Release 2.1.9

Aug 05, 2018

1	Status	3
2	Relevant Background Information and Pre-Requisites	5
3	Contents:	7

The Roboy Dialog System (RDS) is a sophisticated software module representing the cognitive capabilities of the humanoid anthropomorphic robot Roboy. The goal of the project is to implement dialog routines and knowledge extraction for a realistic human-like conversation flow which is achieved by utilizing various behaviour models represented by the State Machine (RDSM) finite automaton defined via a certain Roboy Personality description (file). Within the particular conversation flow stages, the behavioural variability is obtained by extending and redefining the common RDSM State to produce a certain social interaction. The RDSM State both as actor and as reactor regarding the internally formulated output and externally acquired input. For example, the voice input is processed via the Listening Device -> Speech-to-Text -> Analyzers & Semantic Parser -> Linguistics package -> (InferenceEngine) -> State sequence. The RDS in- and output is not limited to speech. It also supports functionalities like emotional in/output, acting on the actual robby and being deployed on a multitude of communication channels for a broader interaction audience.

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 answers questions about facts
- Roboy recognizes once someone saying one's name
- Roboy asks questions about people he meets
- Roboy stores and retrieves the information about people he meets
- Roboy stores and retrieves the information about himself
- Roboy Dialog is deployable on multi-user communication channels
- Roboy plays the Akinator game
- Roboy can understand snapchat filters to be applied on people's faces

In development:

- Roboy updates the information (name, occupation, ect.) about people he meets
- Roboy updates the information about himself
- Roboy recognizes the intent behind an asked questions (age, creator, capabilities etc.)
- Roboy analyzes emotion and responds accordingly

Relevant Background Information and Pre-Requisites

A User should be familiar with:

- Roboy Personality Files
- Roboy Context
- Roboy Memory Module
- Roboy Semantic Parser Module
- ROS

A Developer should be familiar with:

- Roboy Personality Files
- Roboy Context
- Roboy Memory Module
- Roboy Communication Protocol
- Roboy Semantic Parser Module
- Java programming language
- Maven automation tool
- ROS
- rosjava
- Sockets

Reading list for a User:

- [rosjava Documentation](#)

Reading list for a Developer:

- [Java Documentation](#)
- [Maven Documentation](#)

- [rosjava Documentation](#)
- [Roboy Memory Module Documentation](#)

3.1 Installation

In order to install the dialog system it needs to be cloned from github: `git clone https://github.com/Roboy/roboy_dialog` Also, a working installation of Apache Maven is necessary. Please refer to the `README.md` for further installation instructions.

Changing the default configuration, for example the input/output is explained in *Configuration*.

3.2 Getting started

3.2.1 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.states.definitions.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`).

3.2.2 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.

3.2.3 How to extend it?

If you do not know how the system works please refer the “System architecture” section. If you know the part you want to extend or are sure you do not need to understand the detailed functionality of the overall system you may directly start with the “Developer manual” section. The *Code structure overview* section is a good place to start.

3.3 Configuration

3.3.1 Editing the Config File

Dialog System is configured using the `config.properties` file in the root of the project.

ROS configuration

Dialog outsources many tasks to other modules implemented in Python or C++ as ROS packages. In the config file you can enable/disable ROS modules, choose which packages to use, and set the `ROS_MASTER_URI`.

Available ROS packages are:

- `robey_gnlp` (generative model for answer generation)
- `robey_memory` (Neo4j graph-based memory)
- `robey_speech_synthesis` (text to speech using Cerevoice)
- `robey_speech_recognition` (speech to text using Bing Speech API)
- `robey_audio` (audio source localization)
- `robey_vision` (face recognition & object classification and localization)
- `robey_face` (triggers emotions)
- `robey_filters` (triggers snapchat-filters)

Example ROS config:

```
ROS_ENABLED: true
ROS_MASTER_IP: 10.183.49.162
ROS_ACTIVE_PKGS:
- robey_memory
- robey_speech_synthesis
```

Inputs and Outputs

A developer can choose how to interact with the dialog system. For example, for debugging purposes there are command line input and output. Importantly, there can be only one input, but many outputs.

Available inputs are:

- `cmdline`
- `udp` (listens for incoming udp packets in the port specified below)
- `bing` (requires Internet connection and the `robey_speech_recognition` ROS package)
- `telegram` (requires Internet connection and a prepared telegram bot, see 1. Getting Started for more details. For the standard usecase, telegram should be set as both, in- and output.)

Arbitraty of the following outputs can be used simultaneously at the runtime::

- `cmdline`
- `cerevoice` (requires `robey_speech_synthesis` ROS package)
- `ibm` (uses IBM Bluemix, requires Internet connection, user & pass configured below)
- `emotions` (requires `robey_face` ROS package)
- `udp` (sends packets on the port configure below)
- `telegram` (requires Internet connection and a prepared telegram bot, see [Installation](#) for more information. For the standard usecase, telegram should be set as both, in- and output.)

Example IO config:

```
INPUT: cmdline
OUTPUTS:
- cmdline
- ibm
- cerevoice
```

Additional configuration from the “Utilities” paragraph may be required.

System behaviour flags

Debug flag for en/disabling debug specific behaviour:

```
DEBUG = false
```

Demo mode flag for en/disabling demo mode. This is for fairs and such where one would prepare the system for showing whilst giving a talk.

```
DEMO_MODE: false
```

Infinite repetition flag: For input that require a single instance of the dialog system (like command line or on the robey). En/disables beginning a new conversation after one has ended or ending the dialog system when the conversation has ended.

```
INFINITE_REPETITION: true
```

Personality

Here you specify the state machine description store in the JSON file containing personality, i.e. states and transitions between them:

```
PERSONALITY_FILE: "resources/personalityFiles/OrdinaryPersonality.json"
```

Utilities

Configure third party communication ports, credentials, etc.

UDP in-output

```
UDP_IN_SOCKET: 55555
UDP_OUT_SOCKET: 55556
UDP_HOST_ADDRESS: 127.0.0.1
```

IBM Watson text-to-speech

```
IBM_TTS_USER: x
IBM_TTS_PASS: x
```

Telegram JSON-File path (see *JSON Resources*)

```
TELEGRAM_API_TOKENS_FILE: "/path/to/example.json"
```

3.3.2 Configuring a telegram bot

If you'd like to use the telegram in- or output registering your own bot is necessary. Please proceed as follows:

1. Register a bot as described on the [telegram website](#).
2. Place your telegram-bot authentication token in a JSON-File as described in *JSON Resources*.
3. Configure the Dialog System to use your file and to interact with the world via telegram as described above.

3.4 Troubleshooting

3.4.1 Dialog Specific Exceptions

Exception in thread "main" java.lang.OutOfMemoryError: GC overhead limit exceeded

This means that Java does not have enough heap space to execute. One can resolve this by allowing java to use more heap space.

If you are running the class via Maven, add `export MAVEN_OPTS="-Xmx6g"` to your `.bashrc` or equivalent file.

If you are running via IntelliJ and running `DialogSystem` as an application, add `-Xmx6g` to the VM Options. (Likely a similar process for other IDEs)

Note: 6g can be replaced with a larger/smaller number. The project has been confirmed to work with 3000 mb (3000m) as one's parameter.

Gnutls Handshake Fails

If you get an error similar to that below, it means that your internet is working. Please check that your internet connection works.

```
--- exec-maven-plugin:1.6.0:exec (Pull dependencies) @ roboy-parser-nonmaven-deps ---
===== Downloading roboy: Roboy extra utilities (need to compile)
Cloning into 'fig'...
fatal: unable to access 'https://github.com/percyliang/fig/': gnutls_handshake()
↳ failed: Error in the pull function.
[ERROR] Command execution failed.
      org.apache.commons.exec.ExecuteException: Process exited with an error: 1
↳ (Exit value: 1)
      at org.apache.commons.exec.DefaultExecutor.executeInternal(DefaultExecutor.
↳ java:404)

      .....

      at org.codehaus.plexus.classworlds.launcher.Launcher.main(Launcher.java:356)
[INFO] -----
[INFO] Reactor Summary:
[INFO]
[INFO] Sempre External Dependencies ..... FAILURE [ 0.892 s]
[INFO] Roboy Semantic Parser ..... SKIPPED
[INFO] Roboy Semantic Parser Modules ..... SKIPPED
[INFO] Roboy Memory ..... SKIPPED
[INFO] Roboy Dialog System ..... SKIPPED
[INFO] Roboy Dialog System Modules ..... SKIPPED
[ERROR] Failed to execute goal org.codehaus.mojo:exec-maven-plugin:1.6.0:exec (Pull
↳ dependencies) on project roboy-parser-nonmaven-deps: Command execution failed.
↳ Process exited with an error: 1 (Exit value: 1) -> [Help 1]
```

Building Sempre External Dependencies 1.0.0 Fails to Build

If this comes up in combination with a `java.lang.ClassNotFoundException: roboy.dialog.DialogSystem`, then you are likely using the old maven command to build the project. Please use the one found on the installation page.

ContextIntegrationTest Fails

ContextIntegrationTest should have been deprecated and ignored during the testing phase in Maven. If for some reason you have this error, make sure that you are on the latest version of your branch.

3.4.2 Memory Specific Exceptions

Memory has it's own [page](#) regarding exceptions.

3.4.3 Github

If you encounter an error that has not been addressed, feel free to check the issues section of [Dialog and NLU](#), as well as that of [Memory](#), on Github. There, one can check if the issue has been already been reported and if there is a fix. If you still can't find an answer to your question, create a Github issue, so that the dev team is notified and can help.

3.5 Architecture overview

The Roboy Dialog System consists of three interacting parts: The Dialog Systems **'core'**, which provides infrastructure to each conversation the Dialog System can have with a user and overall management of the Dialog System, **'conversations'**, which are threads handling per-user interaction and are embedded into the Dialog System and **'world interfaces'**, which are services and interfaces over which the Roboy Dialog System may communicate with users.

In the figure you may see a simplified overview of the Dialog System. Below that, the general functionality of the system is explained. For details see the specific sub-pages of one of the three parts. Although this is a description of the systems architecture, we will mention implementation specific details for easier understanding.

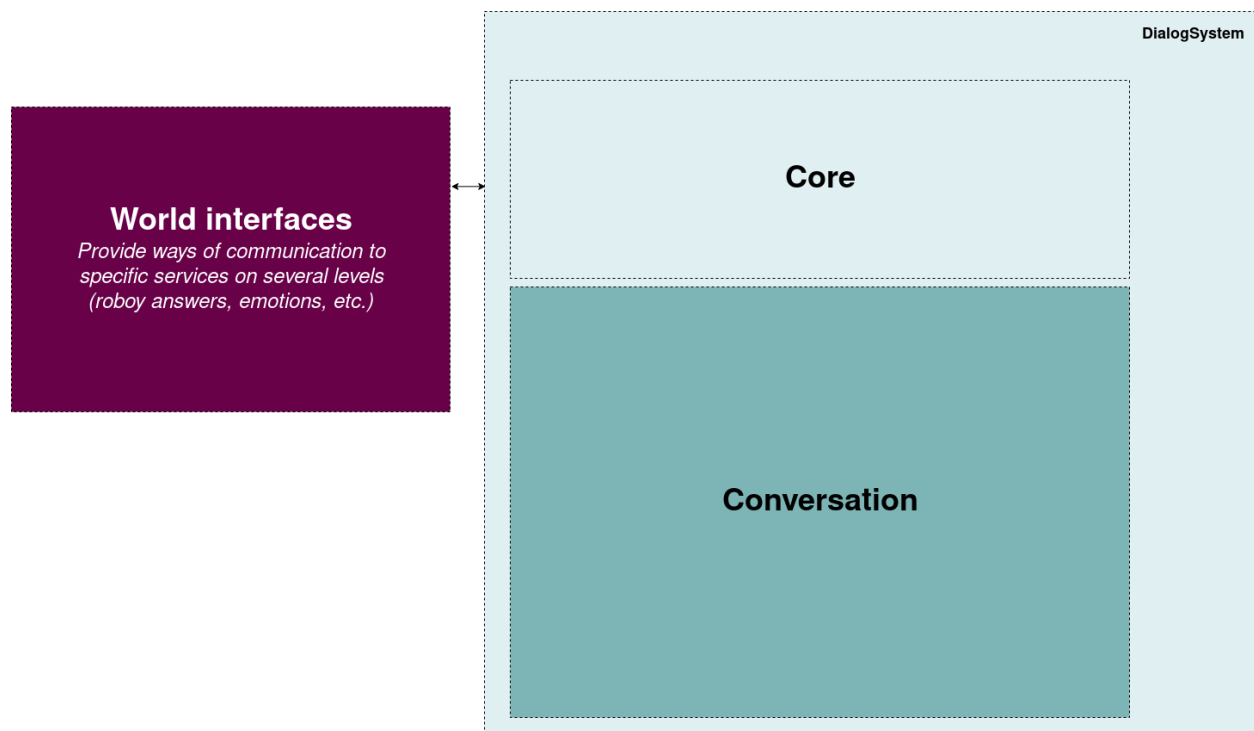


Fig. 1: [Click here](#) in order to see the detailed overview partially provided within the subsections.

3.5.1 Conversation flow

In order to understand how the Dialog System works, a basic lifecycle of a *Conversation* is drawn here exemplary:

When the Dialog System is started, the *ConversationManager* initializes the Dialog Systems *Memory*, the *NLU* and the configured *InputDevice*. The *InputDevice* then starts listening to it's service.

Input from an unknown user will be handled by the *InputDevice*. Typically, it should make the *ConversationManager* spawn a new *Conversation* and then direct the users input to this new *Conversation*.

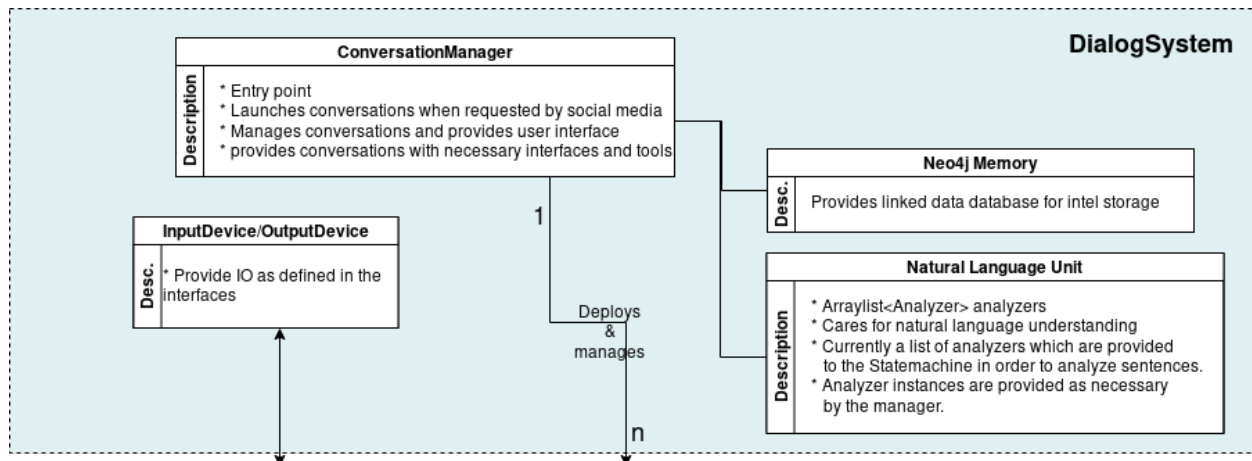
Input into a running conversation will be taken by the waiting *Conversation* thread and then processed using the *NLU*. This processing includes interpreting it, invoking a state machine step and possibly reacting to the input depending on the new state of the dialog, the input itself and possible information from the Dialog Systems *memory* or the *Context*. Generated output will then be delivered to the user via an *OutputDevice*.

The end of a Conversation will usually be triggered by the *StateMachine*. If this happens the Dialog System will take care of cleaning up the *Conversation* and the data it owned.

3.6 The Dialog System Core

The Dialog System Core takes care of spawning, deleting and managing *Conversations*, initializing and providing all necessary static core features and services like the memory, the *NLU*, *StateMachine*, etc. to *Conversations*.

At it's heart is the *ConversationManager*, which is accompanied by the external services of *Memory* and *NLU* and the *InputDevices* and *OutputDevices* which provide interfaces to communication channels which we call *World Interfaces*, since the Roboy Dialog System may interface the world through these.



3.6.1 ConversationManager

The `roboy.dialog.ConversationManager` is the central managing node for the Dialog System. It is the entry point of the program. When `roboy.dialog.ConversationManager.main()` is launched, it initializes all necessary modules and then enters control-mode, where it assumes a behaviour depending on the selected *InputDevice* in order to provide control over the *Conversations*.

During the initialization phase all components that are included in the Dialog System core are initialized. This especially includes the *NLU* (in form of a queue of several `roboy.linguistics.sentenceanalysis.Analyzer` classes) and *world interface* specific initializations (*world interfaces* may for different reasons need a centralized initialization separate from the *I/O* devices which are used to communicate with them).

In order to manage Conversations it stores them by UUID, which is in the format "[world-interface-name]-[uuid-from-world-interface]", for example "telegram-aaaaaaaaaaaa". Then it provides an API consisting of methods that allow controlling the *Conversation* threads if you know their uuid. It allows creating and spawning new *Conversations* and restarting, stopping and pausing existing ones. Also it allows deregistering *Conversations* so they can properly be destroyed when they are over.

3.6.2 InputDevice & OutputDevice

The `roboy.dialog.io.InputDevice` and `roboy.dialog.OutputDevice` classes are interfaces that may be used to provide access to *World Interfaces* to the Roboy Dialog System. Their actual behaviour depends on the architecture of the system that should be interfaced.

In order for input to be handled by the Conversations, an implementation of `roboy.dialog.io.InputDevice` must provide a public `Input listen()` method. A *Conversation Thread* will call this function in order to receive a new `roboy.dialog.io.Input`. If no new input has been generated by the user, the thread should wait inside the `listen()` function until a new input is generated.

To enable a Conversation to send output to a channel, an implementation of `roboy.dialog.io.OutputDevice` must provide a public `void act(List<Action> actions)` which should process the different types of `roboy.dialog.action.Action` according to the capabilities of the communication channel.

3.6.3 NLU (Semantic Parser)

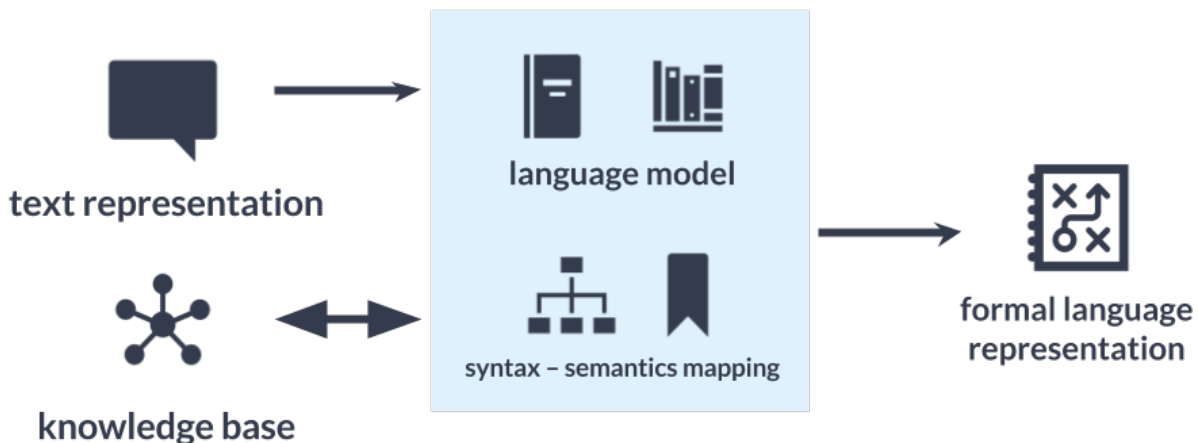
The NLU submodule is used to translate text inputs into formal semantic representations. This allows for capturing the semantic intent behind a statement or question, and using knowledge bases to translate formal question representations into answers.

The `roboy_parser` NLU module is based on *SEMPRE* <<http://nlp.stanford.edu/software/semprer/>>. It is currently being modified to fulfill Roboy Dialog system needs.

Architecture

Semantic parser is based on a language model and NLP algorithms that apply rules to utterances to translate them to semantic representations. Language model consists of: - set of grammar rules, - lexicon, - training dataset.

General architecture can be seen on the diagram below.



For more information please refer to *Natural Language Understanding (nlu/roboy_parser)*.

Linguistic Analyzers

Linguistic analyzers (derived from `de.robey.linguistics.sentenceanalysis.Analyzer`) process user utterances to provide some information (SentenceType, Tokens, Emotion etc.) for Interpretation (`de.robey.linguistics.sentenceanalysis.Interpretation`).

For example, `EmotionAnalyzer` (`de.robey.linguistics.sentenceanalysis.EmotionAnalyzer`) processes user utterances and sets emotion variable of `Interpretation` (`de.robey.linguistics.sentenceanalysis.Interpretation`) with related `RoboyEmotion` (`de.robey.dialog.emotions.RoboyEmotion`) value, by this way `Interpretation` can be checked at `StateBasedPersonality` (`de.robey.dialog.personality.StateBasedPersonality`) for special treatment (responding according to emotion or responding with `EmotionAction` etc.)

3.6.4 Memory (The memory module)

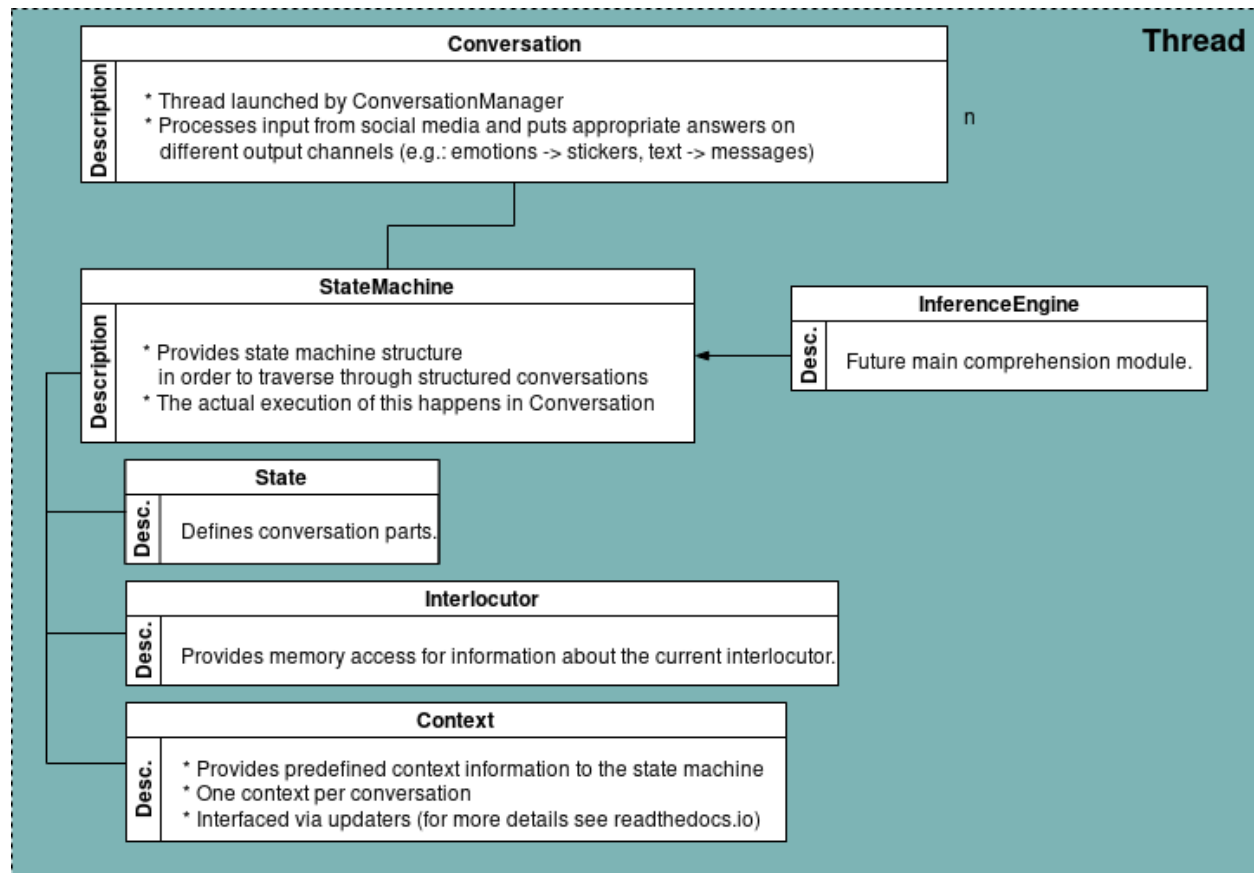
For more in-depth information please refer to [Roboy Memory](#).

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.

3.7 Conversation

A *Conversation* is a concept for everything necessary for conducting a dialog with a single counterpart. It's task is traversing through the different states of the interaction with the user to provide a maximally human-like experience. It receives Input through its *InputDevice* and uses this Input in combination with the *InferenceEngine* and the *StateMachine* to traverse through the correct *States* as defined in the configured *Personality*. If it needs or learns information about the counterpart of the dialog, it uses the *Interlocutor* (see [Roboy Memory](#)) to access the Dialog Systems *Memory*. If it needs to store or inquire information about the environment it is in it uses its *Context* for this.



3.7.1 StateMachine

The `robey.dialog.DialogStateMachine` is the unpopulated base for `robey.dialog.StateBasedPersonality`. It is unique to each *Conversation*, as each dialog usually is in a different state. It holds the *InferenceEngine* and the *Context* for a *Conversation*, as it should be the only part of it, accessing these. Its task is to handle traversing through the *States* of a *Personality* whereas a *StateBasedPersonality* enhances the features of the *DialogStateMachine* by augmenting the state-traversing functionalities with interaction-specific non-state behaviour like detecting the end of a conversation.

3.7.2 Personality and states

Overview

To enable a natural way of communication, Roboy's Dialog System implements a flexible architecture using different personalities defined in **personality files**. Each file represents a **state machine** and defines transitions between 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.

Personality

A personality defines how Roboy reacts to every given situation. Different personalities are meant to be used in different situations, like a more formal or loose one depending on the occasion. Roboy always represents one personality at a time. Personalities are stored in JSON personality files.

During one run-through, the Dialog System uses a single Personality instance (currently implemented in `robey.dialog.personality.StateBasedPersonality`) which is built on top of a state machine. This implementation loads the behaviour from a personality file that stores a representation of a state machine. Additionally, it is possible to define the dialog structure directly from code (as it was done in previous implementations).

As the conversation goes on, the state machine will move from one state to another consuming inputs and producing outputs. The outputs are always defined by the current active state.

State

A state contains logic to control a small part of the conversation. It is a class that extends `robey.dialog.states.definitions.State` and implements three functions: `act()`, `react()` and `getNextState()`.

State's activity can be divided into three stages. First, 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 (see semantic parser), the `react()` method completes the current state's actions. Finally, Roboy picks a transition to the next state defined by the `getNextState()` method of the current state.

State Output

The `act()` and `react()` functions return a `State.Output` object. This object defines what actions Roboy should do at this point of time. Most important actions include:

- say a phrase
- say nothing
- end the conversation and optionally say a few last words

The Output objects are created using static factory functions inside `act()` or `react()` in a very simple way. For example, if Roboy should react with a phrase, the `react()` function could be implemented like this: `return Output.say("some text here")`. Here, `Output.say` is the static factory function that creates an Output object.

To improve the dialog flow, you can add segues to the Output objects using `outputObj.setSegue()`. A segue is a smooth transition from one topic to the next. Or you can add emotions to the Output objects using `outputObj.setEmotion()`. An emotion is used for creating emotional actions which can be shown at Roboy's face by `FaceAction` or at Telegram chats by `EmotionAction` using stickers.

State Transitions

A state can have any number of transitions to other states. Every transition has a name (like "next" or "error"). 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 two outgoing transitions: "gotYes", "gotNo" and "askAgain" (if the reply is not "yes/no").

When designing a new state, the transition names are defined first. The transition name should describe a condition and not another state. For example, a good name would be "knownPerson" (take this transition when you meet a known person) or "greetingDetected" (take this transition when you hear a greeting). In this case, the name only defines a condition and allows the transition to point to any state. In contrary, a bad name would be "goToQuestionAnsweringState" because it implies that no other state than `QuestionAnsweringState` should be attached to this transition. This breaks modularity.

Once the state is implemented, the connections between states are defined in the personality file. At run time the state machine loads the file and initializes the transitions to point to correct states. During the implementation, the destination state can be retrieved by the transition name using `getTransition(transitionName)`.

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. If `getNextState()` returns no next state (`null`), the conversation ends immediately.

Fallback States

Fallbacks are classes that handle unpredicted or unexpected input. A fallback can be attached to any state that expects inputs that it cannot deal with. In the case this state doesn't know how to react to an utterance, it can return `Output.useFallback()` from the `react()` function. The state machine will query the fallback in this case. This concept helps to keep the states simple and reduce the dependencies between them. When implementing the `react()` function of a new state, it is sufficient to detect unknown input and return `Output.useFallback()`.

In the current Dialog System, we use special states to implement the fallback functionality. A fallback state never becomes active so only the `react()` function has to be implemented. This function will be called if the active state returned `Output.useFallback()`.

State Parameters

Sometimes you want to pass parameters to the states, for example define a path to a file that contains some data. Parameters are defined inside the personality file. Each parameter has a name and a string value. When a state is created, the state machine passes all parameters from the file to the state constructor. Therefore, every state sub-class should have a constructor that accepts parameters matching the constructor of the `State` class.

During runtime, state objects can access the parameters using the `getParameters()` function which returns a `StateParameters` object. This object contains parameters from the personality file as well as references to `StateMachine`, `RosMainNode` and `Neo4jMemoryInterface` in case you need them.

State Interface

When you create a new personality file you might forget to define important transitions and provide required parameters to some states. To prevent long debugging and find errors faster you can define an interface for every state. The interface describes:

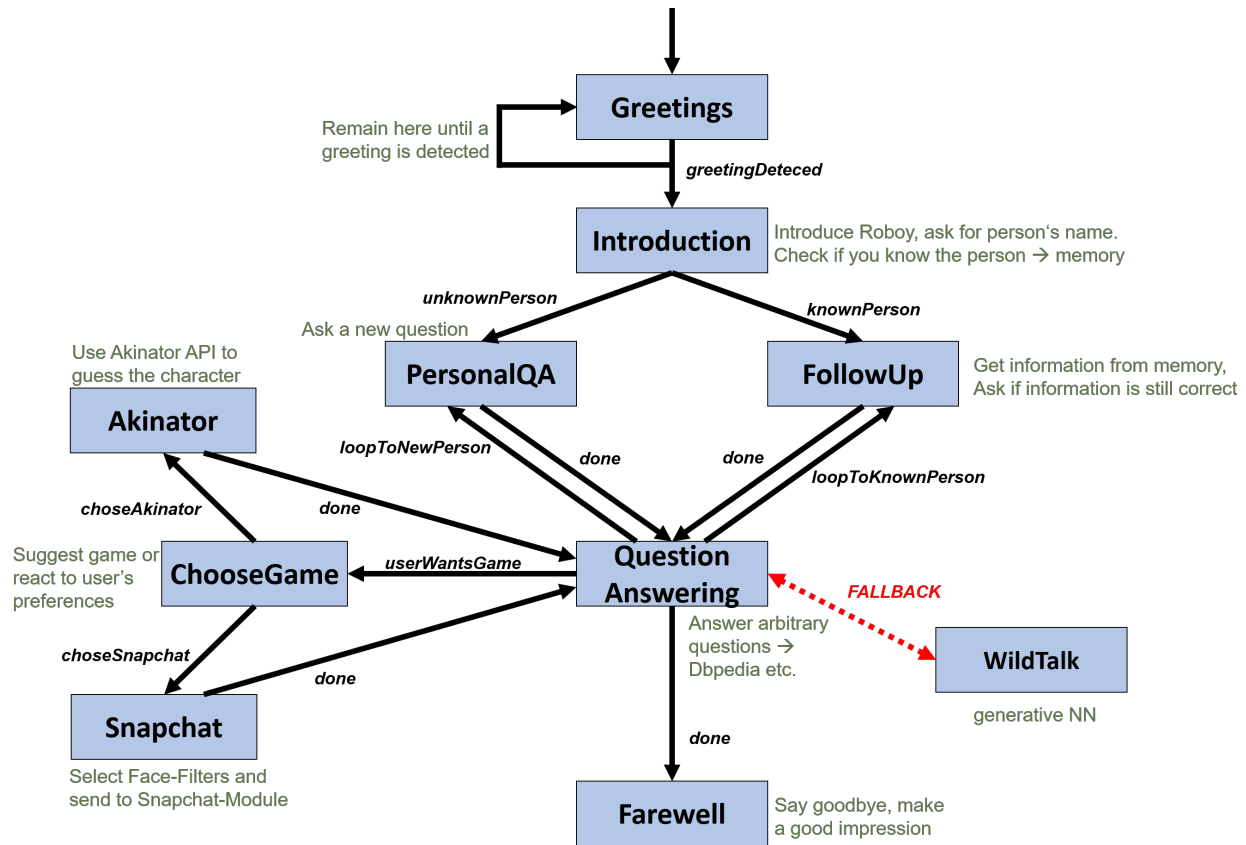
- transitions that have to be set
- parameters that has to be provided
- whether a fallback is required for this state

After the personality file was loaded and the state machine was initialized, the dialog system will check if all states have everything they define in the state interface.

For every state, its interface is implemented by overriding three functions: `getRequiredTransitionNames()`, `isFallbackRequired()` and `getRequiredParameterNames()`. Note, that you don't have to override those functions if your state has no specific requirements.

Current standard Personality

Current standard personality is used to interact with a single person. After Roboy hears a greeting and learns the name of the person, he will ask a few personal questions and answer some general questions about himself or the environment. He will also occasionally suggest to play a game.



Overview over Implemented States

PassiveGreetingsState: Roboy is listening until a greeting or his name is detected (passive state to start a conversation).

IntroductionState: Roboy asks the interlocutor for his name, decides if the person is known and takes one of two transitions: knownPerson or newPerson.

PIAState (PersonalInformationAskingState): Roboy asks one of the personal questions (like ‘Where do you live?’) and updates facts in Memory.

FUASState (FollowUpAskingState): Roboy asks if the known facts are still up to date (like ‘Do you still live in XY?’). This state is only entered if there are some known facts about the active interlocutor.

QuestionAnsweringState: Roboy answers questions about itself or some general questions. Answers are provided by the parser (from sources like DBpedia) or the Memory.

WildTalkState: This fallback state will query the deep learning generative model over ROS to create a reply for any situation.

ChooseGameState: Roboy suggests a game by random choice or let the interlocutor select one.

GamingTwentyQuestionsState: Roboy plays tries to guess a character the interlocutor is thinking of by asking yes-no-questions.

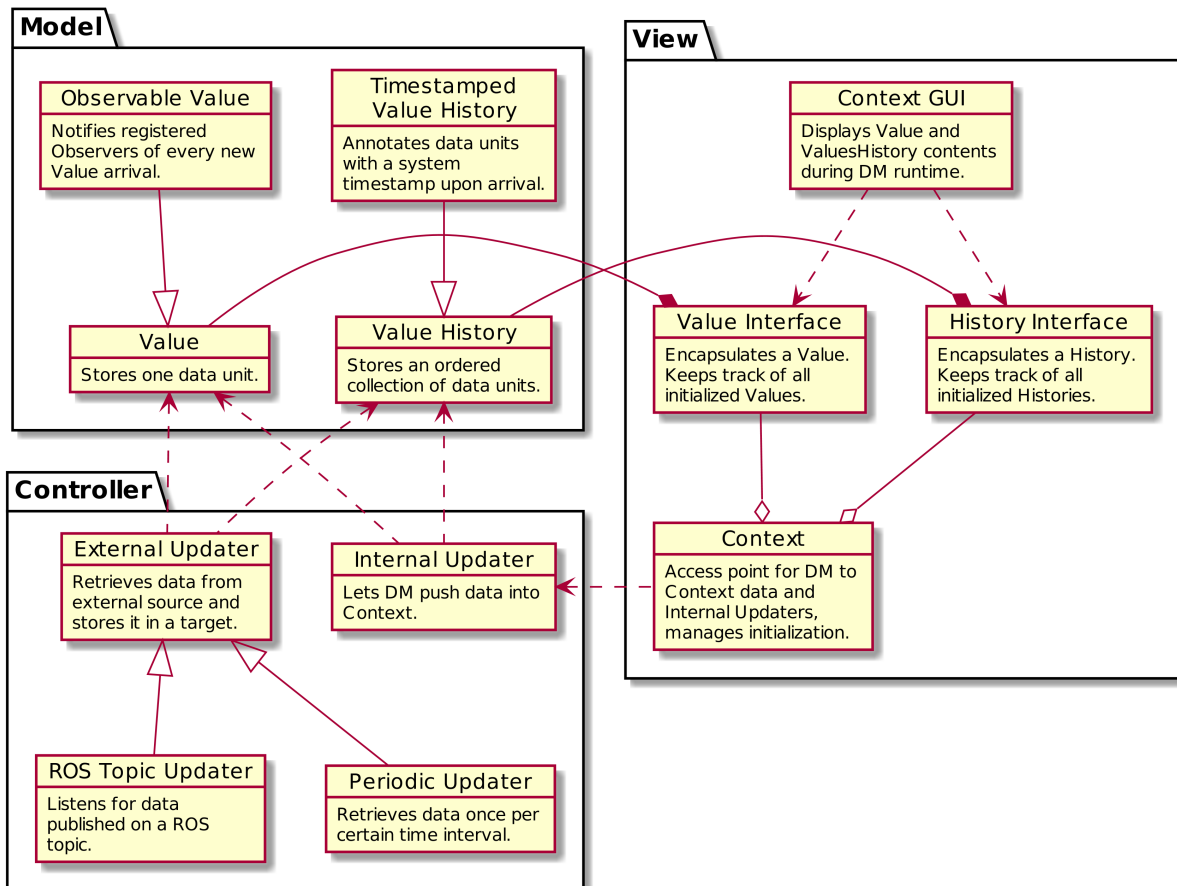
GamingSnapchatState: Roboy suggests or the person can choose Snapchat-Filters which will then be sent over ROS to the snapchat-module.

FarewellState: Roboy ends the conversation after a few statements.

3.7.3 Context

The goal of `robey.context.Context` is to collect information about Roboy's or a conversation's environment and state. It is a per-dialogue structure and will only store contemporary information which will be lost when the conversation. If the Dialog System encounters information about this *Conversations* context, for example where this happens, a *Personality* might store this information here. Also external services may provide contextual information through the *Context*. This information can be used by the dialog manager and also to react upon situations that match certain conditions, such as turning the head of the Roboy when the Interlocutor moves or the last time a message from a user was received in a social-media setting.

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. These may be accessed through `Value Interface` and `History Interface` implementations and may be changed through `Updater` implementations. For more information see [Using the Context](#).

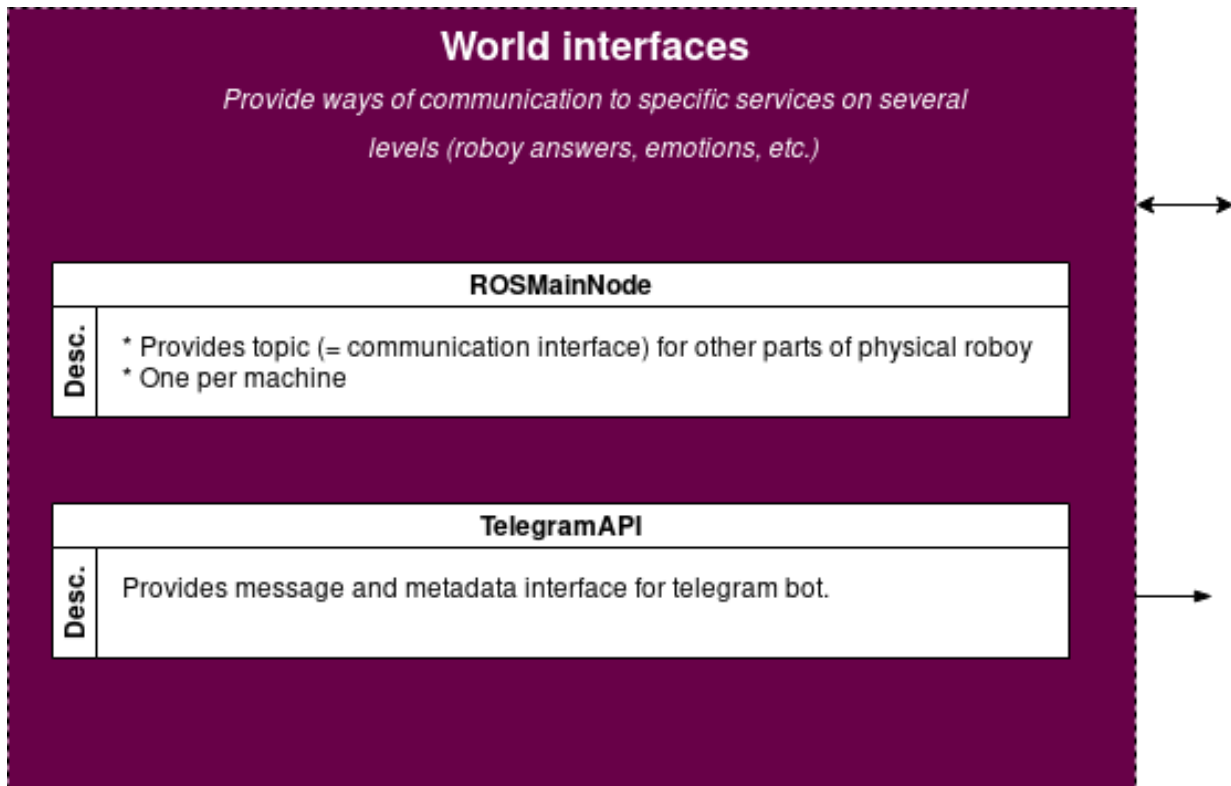
3.7.4 Inference engine

EXPERIMENTAL FUNCTIONALITY

The Inference Engine is one of the main future components of Roboy Dialog System. Its main task is to process the data obtained from various analyzers and parsers to successfully infer the expected set of actions and retrieve the meaningful bits of information as well as ground the references from available ontologies and external sources.

3.8 World Interfaces

A world interace is any external service, device, etc. that enables the Dialog System to interact with the world outside of the dialog system. Interaction with a *World Interface* is provided to a conversation via *InputDevices* and *OutputDevices*. Some need to be initialized or controlled which would be done by the *ConversationManager*. This page is a list of currently supported *World Interfaces* and provides information on how to use them and how they work.



3.8.1 ROS

Note: Robot interface

The Robot Operating System (ROS) provides an interface so the dialog manager may be deployed as the brain of robey. It provides a way to communicate with the actual hardware and using its many ways of acting on and sensing the world as in- and outputs.

The memory, vision, emotion, speech, generative model and middleware communication is carried out through Ros-MainNode object which implements AbstractNodeMain (inheriting NodeListener of rosjava) and offering the control through the following important methods:

- onStart
- onShutdown

- `onShutdownComplete`
- `onError`

Currently, it also provides with the next custom methods:

- `SynthesizeSpeech`
- `RecognizeSpeech`
- `GenerateAnswer`
- `ShowEmotion`
- `DetectIntent`
- `addListener`
- `waitForLatchUnlock`

3.8.2 Cerevoice

Note: I/O Device

Choose `cerevoice` in- and output in `config.properties`.

Cerevoice is a software for speech-to-text and text-to-speech processing. An external cerevoice software instance must be provided in order for the Dialog System to use this.

3.8.3 UDP

Note: I/O Device

Choose `udp` in- and output in `config.properties`.

The dialog system may receive and send in- and output that needs no further processing through a udp port configured in `config.properties`.

3.8.4 Bing

Note: Input Device

Choose `bing` input in `config.properties`.

Bing speech-to-text processing. Requires internet connection and the `robey_speech_recognition` ROS package.

3.8.5 Command line

Note: I/O Device

Choose `cmd` in- and output in `config.properties`.

In- and output via the command line in which the Dialog System is launched. Pure text-based in- and output.

3.8.6 Telegram

Note: I/O Device

Choose `telegram` in- and output in `config.properties`.

Use a telegram bot to interface with the world. See [Configuring a telegram bot](#) for more information.

If the dialog system has been configured to use telegram as input world interface, the command line switches into control mode

- [Deprecated] `CypherMemoryQuery`
- [Deprecated] `DeleteMemoryQuery`
- [Deprecated] `GetMemoryQuery`
- [Deprecated] `UpdateMemoryQuery`
- [Deprecated] `CreateMemoryQuery`

3.9 Documentation Guidelines

If you want to extend this documentation, this page provides help on choosing where to extend it.

3.9.1 User manual

Note: The user manual section is the place where everything a **user** of dialog system needs to read.

All information necessary for someone deploying the dialog system in order to have a humanlike interaction service goes here. This incorporates information or references to them on how to install (especially this information should be in the respective README and just be referenced in this documentation), configure and run the dialog system. Additionally this section includes short explanations and references to development-specific stuff typical users might need like how to build a personality for their usecase.

3.9.2 System architecture

Note: The system architecture section is where developers go to understand the dialog system.

This section includes conceptual information about the dialog system while mentioning code where it fits in order to prepare the reader to explore the code while understanding the dialog system.

3.9.3 Developer manual

Note: The developer manual section is where developers go in order to learn about or look up specific implementation details while developing.

This section contains implementation specific details that need additional explanation and general information about the project structure (where to find what, etc.). If extending the dialog system in a specific place requires multiple steps all over the dialog system, please add a tutorial for this. (For example `roboy.io.InputDevice` and `roboy.io.OutputDevice`)

3.10 Development tutorials

This page is a collection of useful tutorials if you want to develop or enhance parts of the Dialog System.

3.10.1 Changing the dialog systems behaviour during a conversation

Adding a New State

Roboy often visits different events and you might want him to say something specific, for example mention a company or a sponsor. One way to do this would be to modify an existing state. However, these changes are often discarded as you still want to have the old behaviour. There is a better way: create a new custom state specifically for your needs.

In this tutorial you will learn how to design and implement a new state. To keep everything simple, the state will just ask a yes-no question and listen to the answer. Based on the answer, you will pick one of two replies and choose one of two transitions.

Do you know math?

Let's start! We are going to add a state that tests whether the interlocutor (person speaking to Roboy) knows some basic math. First, create a new class named `DoYouKnowMathState` that extends from `roboy.dialog.states.definitions.State`:

```
// inside DoYouKnowMathState.java

public class DoYouKnowMathState extends State {

}
```

Your IDE will notify you that three functions (`act()`, `react(...)` and `getNextState()`) have to be implemented. Let's add them:

```
// inside DoYouKnowMathState.java

@Override
public Output act() {
    return null;
}

@Override
public Output react(Interpretation input) {
```

(continues on next page)

(continued from previous page)

```

    return null;
}

@Override
public State getNextState() {
    return null;
}

```

Additionally, we need a special constructor and a new variable to store the next state for later:

```

// inside DoYouKnowMathState.java

private State next;

public DoYouKnowMathState(String stateIdentifier, StateParameters params) {
    super(stateIdentifier, params);
}

```

Now, we can write some logic and define what our new state should do. The `act()` function is always executed first. In our case, we want to ask a simple question. Replace `return null;` inside `act()` with following:

```

// inside public Output act()

return Output.say("What is 2 plus 2?");

```

The interlocutor's answer will be passed to the `react(...)` function once it is available. Inside, we should check whether the answer is correct and react based on correctness. If we also want to add emotion to our output, `setEmotion()` method can be used. This code is one of the simplest ways to do this:

```

// inside public Output react(Interpretation input)

// get tokens (= single words of the input)
List<String> tokens = input.getTokens();

// check if the answer is correct (simplest version)
if (tokens != null && ((List) tokens).size() > 0 && tokens.get(0).equals("four")) {
    // answer correct
    next = getTransition("personKnowsMath");
    return Output.say("You are good at math!").setEmotion("happiness");
} else {
    // answer incorrect
    next = getTransition("personDoesNotKnowMath");
    return Output.say("Well, 2 plus 2 is 4!").setEmotion("sadness");
}

```

Note a few things here:

- to keep this tutorial simple, we only check whether the first word of the reply equals “four”
- based on reply correctness, we get the next state using `getTransition(<transitionName>)` save it for later
- similarly to `act()`, we define the output with `return Output.say(<stringToSay>);`

Finally, we can implement the last required function `getNextState()` that defines the next state to enter. Inside, we just return the next state that we defined inside `react(...)`:

```
// inside public State getNextState()

return next;
```

That's it, you have just created your first state! Here is how the class should look like:

```
// inside DoYouKnowMathState.java

package roboy.dialog.tutorials.tutorialStates;

import roboy.dialog.states.definitions.State;
import roboy.dialog.states.definitions.StateParameters;
import roboy.linguistics.Linguistics;
import roboy.linguistics.sentenceanalysis.Interpretation;

public class DoYouKnowMathState extends State {

    private State next;

    public DoYouKnowMathState(String stateIdentifier, StateParameters params) {
        super(stateIdentifier, params);
    }

    @Override
    public Output act() {
        return Output.say("What is 2 plus 2?");
    }

    @Override
    public Output react(Interpretation input) {

        // get tokens (= single words of the input)
        List<String> tokens = input.getTokens();

        // check if the answer is correct (simplest version)
        if (tokens != null && ((List) tokens).size() > 0 && tokens.get(0).equals("four
↪")) {
            // answer correct
            next = getTransition("personKnowsMath");
            return Output.say("You are good at math!").setEmotion("happiness");
        } else {
            // answer incorrect
            next = getTransition("personDoesNotKnowMath");
            return Output.say("Well, 2 plus 2 is 4!").setEmotion("sadness");
        }
    }

    @Override
    public State getNextState() {
        return next;
    }
}
```

The newest version of the complete code can be found in `roboy.dialog.tutorials.tutorialStates.DoYouKnowMathState`. Read the *Creating a New Personality* tutorial to learn how to connect your new state with others.

Example output

When using the new state, you could encounter the conversation:

```
[Roboy]: What is 2 plus 2?
[You]:   four
[Roboy]: You are good at math!
```

Or, if you provide a wrong answer:

```
[Roboy]: What is 2 plus 2?
[You]:   one
[Roboy]: Well, 2 plus 2 is 4!
```

To learn more details about states and personalities, refer to *Personality and states*. There, you will find details about state fallbacks, parameters and interfaces, as well as more information about different personalities and more output options.

Creating a New Personality

Roboy's Dialog System can be used in different environments and situations like fairs, conferences, demos or as a chatbot on social networks. For every given situation, Roboy's behaviour should be different. We use personalities to define Roboy's way of talking.

In this tutorial you will learn how to create a new personality. Make sure that you know the basic functionality of states. If you are not familiar with them, read the *Adding a New State* tutorial. General information about personalities can be found on *Personality and states*.

Personalities are defined inside JSON personality files. Each file represents a state machine and defines:

- initial state: state in which Roboy starts the conversation
- transitions: connections between the states and the dialog flow
- fallbacks: backup if a state fails to react to unexpected input

State definition

Every state inside the personality file is defined by a JSON object. Here is an example:

```
{
  "identifier": "MathTest",
  "implementation" : "robey.dialog.tutorials.tutorialStates.DoYouKnowMathState",
  "transitions" : {
    "personKnowsMath" : "Farewell",
    "personDoesNotKnowMath" : "Farewell"
  },
  "comment": "A state that will test your math knowledge."
}
```

We have just defined a state that is called `MathTest`. Every state must have a unique identifier.

The `implementation` property defines which Java class should be used for this state when the Dialog System is running. When the Dialog System loads a personality file, it creates a Java object of the right class for *every* state defined in the file.

It is important to provide the complete class name (including the package) so that the Dialog System can find the right class and instantiate an object of it when loading the file. Special care is needed when doing refactoring. Make sure to change the personality file when you rename a state or move it to a different package!

Next, we have transitions. Here we define the connections between states. You should define all transitions that could be taken by the state implementation. For the `DoYouKnowMathState` we have two of them: `personKnowsMath` and `personDoesNotKnowMath` (look for `getTransition(<transitionName>)` inside the Java code). In the JSON file, the key is the transition name (e.g. `personKnowsMath`) and the value (here `Farewell`) is the identifier of another state in the *same* personality file (do not confuse with Java class names). We will take a look at the definition of the `Farewell` state a bit later.

The `comment` property is optional and will be ignored completely by the Dialog System. It just gives you an option to note some details about the state. There are two additional properties that you can (and sometimes have to) define: `parameters` and `fallback`. We will discuss them later as well.

Now, let's define the `Farewell` state. We will use one of the pre-implemented toy states. The definition looks like this:

```
{
  "identifier": "Farewell",
  "implementation" : "robey.dialog.tutorials.tutorialStates.ToyFarewellState",
  "transitions" : {},
  "comment": "Last state: Tells goodbye, ignores reply, ends the conversation."
}
```

Nothing new here, except that we have no outgoing transitions for this state. This is because the `ToyFarewellState` always ends the conversation and will never take any transition.

Complete personality file

With two states defined, we can now take a look at the complete personality file. All state definitions are stored in the `states` array. Additionally, we define the `initialState` and pass the identifier `MathTest` of our `DoYouKnowMathState`. The complete file looks like this:

```
{
  "comment": "A simple personality that only contains two states (used in tutorial).",
  "initialState": "MathTest",
  "states": [
    {
      "identifier": "MathTest",
      "implementation" : "robey.dialog.tutorials.tutorialStates.DoYouKnowMathState",
      "transitions" : {
        "personKnowsMath" : "Farewell",
        "personDoesNotKnowMath" : "Farewell"
      },
      "comment": "A state that will test your math knowledge."
    },
    {
      "identifier": "Farewell",
      "implementation" : "robey.dialog.tutorials.tutorialStates.ToyFarewellState",
      "transitions" : {},
      "comment": "Last state: Tells goodbye, ignores reply, ends the conversation."
    }
  ]
}
```


This file is stored under `resources/personalityFiles/tutorial/MathTest.json`. You can try running this personality by setting the path (`PERSONALITY_FILE`) in the config file (`config.properties`).

When you create a new personality file you might forget to define important transitions. To find errors faster, you can define the state interface (required transitions, parameters and fallback) for every state when you implement it. While loading the personality file, the Dialog System will check whether the state has everything it needs and warn you if something is missing. Read more about state interfaces on [Personality and states](#).

Fallbacks and parameters

There are two additional properties that you can add to a state definition: `parameters` and `fallback`. Take a look at an example:

```
{
  "identifier": "Intro",
  "implementation": "robey.dialog.tutorials.tutorialStates.ToyIntroState",
  "transitions": {
    "next": "Farewell"
  },
  "fallback": "RandomAnswer"
  "parameters" : {
    "introductionSentence" : "My name is Roboy!"
  }
}
```

Let's take a look at both properties. Here we define `RandomAnswer` (which is an identifier of another state in the same personality file) as the fallback for the state with identifier `Intro`. This means that if `Intro` cannot react to an input, the `RandomAnswer` will be asked instead. The property `parameters` allows you to pass parameters to the state. Each parameter has a name (here `introductionSentence`) and a string value. The state implementation can access the value by the name. Parameters are very useful to pass resource file paths to states. Read more about fallbacks and parameters on [Personality and states](#).

Larger personality

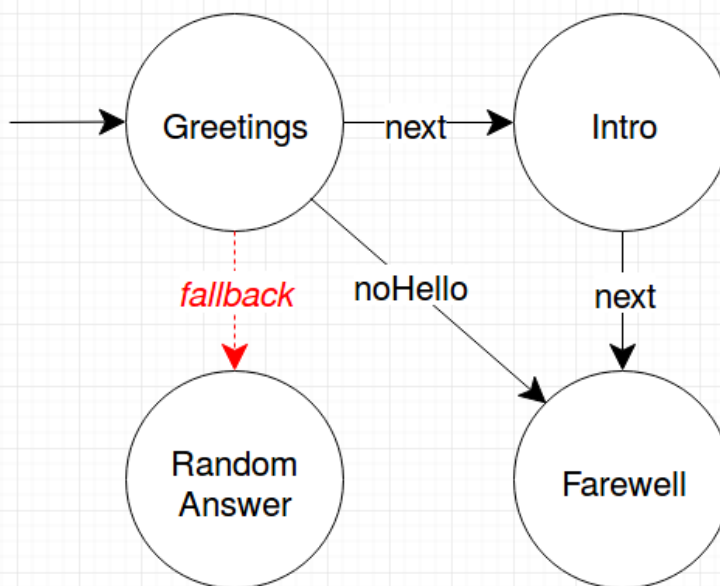
It is not easy to create interesting conversations using only two states (assuming relatively simple states of course). Usually, you will use at least five different states in one conversation. To get some experience in writing personality files, let's create a file that uses four states. Don't worry, you don't have to implement the states here. We will use four already pre-implemented toy states that can be found in the `robey.dialog.tutorials.tutorialStates` package. The final personality should look like this:

As you can see, we have four states that are connected to each other. The names of the transitions are denoted on the arrows. Now, try to write a personality file to represent this personality. Following these steps might be helpful:

- read the Javadoc of every state you will use (`ToyGreetingsState`, `ToyIntroState`, `ToyFarewellState` and `ToyRandomAnswerState`)
- create a new personality file (you might copy `MathTest.json` to have an easier start)
- create four state definitions with different identifiers (`Greetings`, `Intro`, `Farewell` and `RandomAnswer`)
- define the initial state of your personality (`Greetings`)
- define the transitions between the states (note that fallback is not a transition)
- define the fallback for the `Greetings` state
- define required parameters for the `Intro` state (read Javadoc of `ToyIntroState` for details)

Greetings: Say "Hello"; If the reply was a greeting, go to "**next**" state. Otherwise, give no answer (trigger fallback), go to "**noHello**" state.

Intro: Say "My name is Roboy"; Ignore the reply and go to "**next**" state in any case.



RandomAnswer: This state never becomes active and is only used as the fallback. Always reacts with "Roboy is awesome".

Farewell: Say "goodbye" wait for answer and end the conversation.

- save the file in the `resources/peronalityFiles` folder
- edit the `config.properties` file and change `PERSONALITY_FILE` to your path
- try running the Dialog System

If anything goes wrong, you can always take a look at the solution saved in `resources/peronalityFiles/tutorial/ToyStateMachine.json`. Happy personalizing!

Why do we need this complexity?

You might be wondering why such a complex system with all those JSON files is needed. It would be much simpler to define all the states and transitions directly from code, right? Defining everything from code would indeed simplify the refactoring. However, definitions inside personality files have some essential advantages. First, you don't have to recompile the project just to change a personality. Second, in the future, we plan to implement a graphical editor for personalities and therefore need a file format to store the personalities. Using the editor, you will be able to create your own personality with drag & drop and don't have to worry about writing the personality files manually anymore.

While the editor is not implemented yet, we still have good news for you. You *can* define personalities directly from code and don't have to worry about creating a personality file (and updating it while refactoring). This feature is especially useful when writing unit tests for single states or smaller state machines. This tutorial does not cover creating personalities from code but there are good examples in the `roboy.dialog.tutorials.StateMachineExamples` class. Take a look at it if you need to define personalities from code.

Adding New Questions to the State

There exists a list of questions, we may want Roboy to ask in order to acquire new information about people and the environment. It is stored in the resources directory under `sentences/QAList.json` and follows the next JSON structure as given:

```

“FRIEND_OF”: {
  “Q”: [ “Who is your best friend?”, “Have I met any of your friends?”, “Do you have a
        friend whom I have met?”, “Maybe I know some friends of yours. Would you name
        one?”
  ], “A”: {
    “SUCCESS”: [ “Oh, I believe I have met %s they’re nice.”
  ], “FAILURE”: [
    “I don’t think I know them.”
  ]
  }, “FUP”: {
    “Q”: [ “Have you made any new friends, %s?”
  ], “A”: [
    “Oh, I have met %s they’re nice.”
  ]
  }
}

```

Here, we have a set of questions about friends (“FRIEND_OF” intent), so Roboy can learn about friends of the person he is talking to. “SUCCESS” and “FAILURE” are the answers, Roboy will say if the information input was processed successfully or not, respectively. Follow up questions (“FUP”) are the ones that are used to update the information in the future if the questions (“Q”) were already asked.

We can add a new entry there with a new intent. Let it be “LIKE”:

```
“LIKE”: {
    “Q”: [ “What do you like?”
    ], “A”: {
        “SUCCESS”: [ “Me too. I really like %s!”
        ], “FAILURE”: [
            “Well, I do not know what to think about this”
        ]
    }, “FUP”: {
        “Q”: [ “Do you still like, %s?”
        ], “A”: [
            “Maybe, I should consider liking this stuff”
        ]
    }
}
```

Then we have to add a new entry into our local ontology - Neo4jRelationships:

```
public enum Neo4jRelationships {
    ...
    LIKE("LIKE");
    ...
}
```

Go back to your state and inside the act() method implement the following logic:

```
Interlocutor person = getContext().ACTIVE_INTERLOCUTOR.getValue();

RandomList<String> questions = qaValues.getQuestions(Neo4jRelationships.LIKE);
String question = questions.getRandomElement();
return State.Output.say(question);
```

Now, we can ask these newly added questions and later process the answers in the react() method.

Querying the Memory from the Dialog System

Indeed, the newly created state may be the pinnacle of State Machines practice, but it does not yet exploit all of the Roboy Dialog System capabilities, such as the graph database Roboy Memory Module which allows to store and recall information about the environment. For instance, you may want to check whether you belong to the circle of Roboy’s friends.

Every state is bundled with the memory reference inside its parameters, to call the memory you have to access it the following way:

```
Neo4jMemoryInterface memory = getParameters().getMemory();
```

Then you may want to call one of the most used methods, namely, `getById` - which will query the Neo4j database and get the description of the node with the same (unique) ID in JSON format. Roboy's ID is 26.:

```
String requestedObject = getMemory().getById(26);
MemoryNodeModel robby = gson.fromJson(requestedObject, MemoryNodeModel.class);
```

The `MemoryNodeModel` class is the general class which is a model for the nodes stored in Neo4j. It has a label, an ID, parameters and relationships with other nodes denoted by IDs. As soon as we have the Roboy node we can get his friends' IDs like this:

```
ArrayList<Integer> ids = robby.getRelationships(Neo4jRelationships.FRIEND_OF);
```

Then we can proceed with checking Roboy's friends as follows:

```
RandomList<MemoryNodeModel> robbyFriends = new RandomList<>();

if (ids != null && !ids.isEmpty()) {
    try {
        Gson gson = new Gson();
        for (Integer id : ids) {
            String requestedObject = getParameters().getMemory().getById(id);
            robbyFriends.add(gson.fromJson(requestedObject, MemoryNodeModel.class));
        }
    } catch (InterruptedException | IOException e) {
        logger.error("Error on Memory data retrieval: " + e.getMessage());
    }
}
```

Let's check if we are friends with him:

```
if (!robbyFriends.isEmpty()) {
    for (MemoryNodeModel friend : robbyFriends) {
        if (friend.getProperties().get("name").toString() == myName) {
            success = true;
            break;
        }
    }
}
```

However, there exists a special Roboy node class initialized in a specific way like this:

```
Roboy robby = new Roboy(memory);
```

It will retrieve and fill all the data for Roboy from the memory.

Furthermore, we wanted to make it less of miserable routine thus there is a helper function in the `State` superclass, which makes your life much easier:

```
RandomList<MemoryNodeModel> nodes = retrieveNodesFromMemoryByIds(robby.
    ↳getRelationships(Neo4jRelationships.FRIEND_OF));

if (!nodes.isEmpty()) {
    for (MemoryNodeModel node : nodes) {
        if (node.getProperties().get("name").toString() == myName) {
            success = true;
            break;
        }
    }
}
```

(continues on next page)

(continued from previous page)

```
}  
}  
}
```

Creating a Value History / Storing and Updating Values in the Context

See *Using the Context*

Extending the Lexicon and the Grammar

This tutorial explains how to create or change grammar and lexicon used in the semantic parser.

To create your own custom lexicon, you need to create a new file or copy an existing lexicon and add lexemes in the following format:

```
{lexeme:"LEXEME", formula:"FORMULA", type:"TYPE"}
```

where:

- lexeme - is a natural language utterance, e.g., name
- formula - is a semantic representation of the lexeme, e.g., rb:HAS_NAME
- type - is a lexeme type, e.g., NamedEntity, fb:type.any

Additionally, you can also add features in JSON format for map:

```
{lexeme:"name", formula:"rb:HAS_NAME", type:"DataProperty", features:"{feature1:0.5, ↵  
↵feature2:0.3}"}
```

To create your own custom grammar, you need to create a new file or copy existing grammar and add rules in the following format:

```
(rule [Category] ([Expression]) ([Function]))
```

where:

- Category - is a category of rule, for root derivation use \$ROOT
- Expression - is a format of text accepted by the rule expressed in your custom categories or names, e.g., \$PHRASE, \$TOKEN, \$Expr
- Function - semantic function that should be applied to specified pattern, e.g., IdentityFn

Example rules:

```
(rule $ROOT ((what optional) (is optional) $Expr (? optional)) (IdentityFn))  
(rule $Expr ($Expr $Conversion) (JoinFn backward))
```

For in-depth tutorial on expression and function types, refer to original SEMPRES [tutorial](#) or [documentation](#)

To use created files, you need to set the correct parameter in `pom.xml` file. For grammar:

```
-Grammar.inPaths
```

For lexicon:

```
-SimpleLexicon.inPaths
```

Scoring Functions and Knowledge Retrieval

Currently, our semantic parser uses error retrieval mechanism that can be modified in the following steps:

1. Move to package:

```
edu.stanford.nlp.sempre.robey.score
```

2. Implement `edu.stanford.nlp.sempre.robey.score.ScoringFunction` class with `score` method.
3. Add scoring function in constructor of `edu.stanford.nlp.sempre.robey.ErrorRetrieval` class.

1. Move to package:

```
edu.stanford.nlp.sempre.robey.error
```

2. Implement `edu.stanford.nlp.sempre.robey.error.KnowledgeRetriever` class with `analyze` method.
3. Add knowledge retriever in constructor of `edu.stanford.nlp.sempre.robey.ErrorRetrieval` class.

3.10.2 Using the Context

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 `robey.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. Make the value available for the Dialog System by defining a `ValueInterface` in the `Context.java` class, among other class variables. A `ValueInterface` takes two type parameters: the `Value` class created in step 2, and its data type (in our case, `String`). Example: `public final ValueInterface<SampleValue, String> SAMPLE_VALUE = new ValueInterface<>(new SampleValue());`
4. Congratulations, you can now query the new `Value` object! ...but it does not receive any values yet. To change this, see “How to add Updaters?” below.

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. The `contains(V value)` method checks whether an object is currently found in the history - note that `ValueHistories` have size limits, therefore oldest values disappear from the history when new ones are added.

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

1. extend `ValueHistory<>` instead of `Value<>`. If the history should keep more than the default 50 values, override the `getMaxLimit()` method to return your desired limit value.
2. in `Context.java`, create a `HistoryInterface` instead of `ValueInterface`.

How to add Updaters?

New values can only flow into the `Context` over an `Updater` instance. Internal Updaters can be used by the dialog manager to actively add new values. External Updaters run in separate threads and query or listen for new values, for example over a ROS connection.

Updaters only add a single new data unit, relying on the `AbstractValue.updateValue()` method. Thanks to the inheritance chain, you can use an arbitrary `Value` or `ValueHistory` implementation as the target of an updater.

Currently, there are two implementations of an External Updater: `PeriodicUpdater` and `ROSTopicUpdater`.

`PeriodicUpdater` calls an updating method after a certain time interval has passed. To use the periodic updating functionality:

1. Create a class extending `PeriodicUpdater` and implement its `update()` method. It should retrieve the values and finally add them over the `target.updateValue(value)` method call.
2. A constructor is required for the class. Simply match the `PeriodicUpdater` constructor and call `super(target)` within - or use the two-parameter constructor to change the update frequency (by default 1 second).

`ROSTopicUpdater` subscribes itself to a ROS Topic and reacts to messages coming from the topic. To use:

1. Create a class extending `ROSTopicUpdater` and define the `getTargetSubscriber()` method, which will point the updater towards its target ROS topic. The options for the subscriber can be found in the `RosSubscribers.java` class.
2. Implement the `update()` method of the new class. This method will be called whenever a new message is stored in the internal message variable, so it might be enough to just call `target.updateValue(message)`. If the data needs to be extracted from the message first, do it in the `update()` before calling `target.updateValue`.

All External Updaters need to be initialized in the `Context.java` class. To do this:

1. Define the External Updater a private class variable to the `Context.java` class (look for the external updater definition section).
4. If the Updater depends on ROS, add its initialization into the `Context.initializeROS(RosMainNode ros)` method, otherwise add it to the private constructor `Context()`. As the parameter, use the inner value or valueHistory variable from a `ValueInterface` or a `HistoryInterface`.
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.
4. Define the Internal Updater in the `Context.java` class. Initialize the updater within the private `Context()` constructor. For example:

```
public final SampleUpdater SAMPLE_UPDATER; // Define as class variable
SAMPLE_UPDATER = new SampleUpdater(DIALOG_TOPICS.valueHistory); // Initialize
in the constructor
```


3.10.3 Adding generic Input- or OutputDevice

In order to add new `roboy.io.InputDevice` and `roboy.io.OutputDevice` classes, changes in multiple locations are necessary.

1. Implement your `InputDevice` or `OutputDevice` implementation using class `[YOUR CLASSNAME]` extends `InputDevice` (or `OutputDevice`, if you're doing output).
2. If your device needs additional cleaning in order to be destroyed properly, additionally use implements `CleanUp` and implement the `cleanup()` method.
3. Add your devices to `roboy.util.io` in `getInputs()` and `getOutputs()`, so the dialog system may use them if they're chosen in the configuration.
4. Add a (commented) input/output configuration to `config.properties`. (See [Adding new entries to config.properties](#) how to do this)

3.10.4 Social Media Integration

A new InputDevice for a social media

First create a new class in `roboy.io` folder, namely `MySocialMediaInput` that implements from `roboy.io.InputDevice`.

```
// inside MySocialMediaInput.java

public class MySocialMediaInput implements InputDevice {

}
```

One function namely “listen()” has to be implemented. This function is called by a thread and should return a new Input or keep the thread waiting if there isn't any new Input available.

```
@Override
public Input listen() throws InterruptedException, IOException {
    return null;
}
```

Since you will have an `InputDevice` for each user then you need at least a unique identifier for each user right? So each of this unique identifiers should mapped to an `InputDevice`. And since a generic social media input device is structured in a way that a single entry point is necessary, you need a central point to find all our inputdevices. Therefore, create a static hashmap for it as follows.

```
private static final HashMap<String, MySocialMediaInput> inputDevices = new HashMap<>
    < > ();
```

Note: In further steps unique identifier mentioned as uuid

Add a constructor that receives the uuid as parameter

```
// inside MySocialMediaInput.java

public MySocialMediaInput(String uuid) {

    //constructor

}
```

(continues on next page)

(continued from previous page)

```
synchronized(inputDevices){
    inputDevices.put(uuid, this)
}
}
```

At this point, we received the uuid and have a hashmap of each `MySocialMediaInput`. What else we need to implement?: - Return messages as `roboy.io.Input` in the `listen()` method - Receive the messages

Note: The order is actually reversed for the sake of tutorial of course you need to receive messages before you return them.

Let's continue with first one. To return a message we need a message so create a `String` for it right below the `HashMap`.

```
private volatile String message;
```

We need to initialize it in constructor. Add the following into the beginning of constructor.

```
// inside public MySocialMediaInput(String uuid)
this.message = "";
```

Finally finish the listen method

```
// inside MySocialMediaInput.java

public Input listen() throws InterruptedException, IOException {
    Input newInput;
    synchronized(this) {
        while(message.equals("")) {
            try{
                this.wait();
            }
            catch(InterruptedException e){
                if(message == null || message.equals("")) {
                    throw e;
                }
            }
        }
        newInput = new Input(message);
        message = "";
    }
    return newInput;
}
```

Nice, now only thing to worry about is how to receive the message.

Note: There is no `SocialMediaHandler` as template. You should have a handler or any logic that receive the messages from your social media. Then you need to call this function after applied your logic (e.g. wait for a certain time to answer.)

Create a static `onUpdate(Pair<String, String>)` function that will be called from your `SocialMediaHandler` class with pair parameter that consists of the uuid and the message.

```

public static void onUpdate(Pair<String, String> update){
    //get the uuid

    //get the inputdevice

    //assign the message to the input device
}

```

To create the uuid that we discussed before, get the unique identifier from the update. And add a social media name as prefix.

```

//get the uuid

String id = update.getKey();
String uuid = "MySocialMedia-" + id;

```

Note: Why we add a prefix? Because it is possible if there is a same identifier from another social media.

Now we need to get the input device there is an existing one with the uuid.

```

//get the inputdevice

MySocialMediaInput input = inputDevices.get(uuid);
if (input == null){
    try{
        ConversationManager.spawnConversation(uuid);
    }catch(IOException e){
        // do your logging or other error handling stuff
        return;
    }
    input = inputDevices.get(uuid);
}

```

As you can see if there is no inputdevice with respective uuid. `ConversationManager.spawnConversation(uuid)` is used. It magically creates the inputDevice (as well as the Conversation and the magical stuff that you do not need to worry about)

Since you needed to store references to your devices in the hashmap, you need to delete them when a conversation is over so the garbage collector can delete them. Therefore the device needs cleaning. Finally, implement another interface namely `CleanUp` and override its `cleanup()` method.

```

// inside MySocialMediaInput.java

public class MySocialMediaInput implements InputDevice, CleanUp {

    ...

    @override
    public void cleanup() {
        inputDevices.values().remove(this);
    }

}

```

Done! Congratulations, you have just created your social media input device. . But it doesn't work with only input device you also need to an output device for each conversation to send the output.

A new OutputDevice for a social media

You have perfectly working input device for your social media. But that only for receiving messages, we also need to send messages.

Create a new class in `roboy.io` folder namely `MySocialMediaOutput` that implements from `roboy.io.OutputDevice`.

```
// inside MySocialOutput.java

public class MySocialMediaOutput implements OutputDevice {

}
```

You should override a method namely `act` and List of actions as parameter.

```
@Override
public void act(List<Action> actions){
    // handle actions
}
```

Leave the inside of the method empty for now.

As discussed before there is an `OutputDevice` for a user that is communicating with. And a unique identifier that is representing the user for each `OutputDevice`. Again just like our `InputDevice` you need a constructor and `uuid` as parameter.

```
// inside MySocialOutput.java

private String uuid;

public MySocialMediaOutput(String uuid){
    //constructor
    this.uuid = uuid.substring(uuid.indexOf('-')+1);
}
```

Remember the `uuid` in `MySocialMediaInput` was “`MySocialMedia-`”+`id`. Here it is splitted from the original user `id` that will be using for sending message.

Finish the `act` method

```
// handle actions
for(Action a : actions) {
    if (a instanceof SpeechAction) {
        // Normal text message
        String message = ((SpeechAction) a).getText();
        /* SEND THE MESSAGE with your social media handler or directly here the way
↳is up to you */
    }else if (a instanceof EmotionAction) {
        String stickerID = null;
        switch(((EmotionAction) a).getState()){
            case "shy": /*use the method that sends an sticker or emoji or anything
↳that shows emotions, again you can use your social media handler or any other
↳method */
                break;
        }
    }
}
```

Note: In this tutorial, only shy emotion has been used, but there are several emotions you can check `roboy.emotions.RoboyEmotion.java` if you want more!

```
/* */ these comments are not completed you should use your way that is sending a message via social media using the user's id.
```

Now you need to tell the dialog system how to use your new in- and output. Refer to *Adding generic Input- or OutputDevice* in order to tell the dialog system where to find them and how to allow users to activate them. Now rebuild your code, select your Input/OutputDevice in `config.properties` and run it to see the work you have achieved.

Telegram: Handle commands

New inline commands can be handled in `onUpdateReceived` method which is in `TelegramCommunicationHandler` class.

Find the below if code block in `onUpdateReceived`.

```
if(text.startsWith("/")){
    //command
}
```

This block is only checking if the incoming message has a ‘/’ at the beginning of the word, just like all the commands “/start”, “/stop”, “/desired_command”

Let’s try to send a sticker after a command catch. Check if the command is “/like”.

```
//command

if(text == "/like"){
    String stickerId = "CAADAgADOQAD5dCAEOtbFzZ0NKh2Ag"
    sendSticker(chatID, stickerID)
}
```

Note: Each sticker has its own unique id in Telegram.

3.10.5 Adding new entries to `config.properties`

In order to add a new configuration possibility, changes in `config.properties` and `roboy.util.ConfigManager` are necessary. This tutorial will use the boolean example of `DEMO_MODE` so it may be retraced by looking at the existing code.

1. Add a new property to `config.properties`

```
DEMO_MODE: false
```

2. Add your new value and a default initialization to the top of “`roboy.util.ConfigManager`”

```
public static boolean DEMO_MODE = false;
```

3. Read the new value from `config.properties` in the lower part of `roboy.util.ConfigManager`

```
DEMO_MODE = yamlConfig.getBoolean("DEMO_MODE");
```

3.11 Code structure overview

This page gives overview of the project to help navigating the dialog system code.

3.11.1 The project structure

dialog

Contains everything specific to the Dialog System functionalities.

resources/

Resources needed by the dialog system only.

src

integration-test: Contains integration tests for combined testing of several Dialog System units.

test: Contains unit tests for testing isolated Dialog System units.

main

context: Contains the *Context* and all classes only relevant in combination with *Context*.

dialog: Everything necessary to conduct the concept of a dialog: Personalities, Statemachine, Conversations, etc.

emotions: Contains information about possible emotions.

io: Contains Input- and OutputDevices.

linguistics: Contains NLU Wrappers + help classes for them and some very simple linguistic analysis.

logic: Contains deduction and inference.

memory: Contains memory wrappers and processing classes.

ros: Contains ROS wrappers and interfaces.

talk: Contains sentence-building logic.

util: Contains Dialog System wide utilities.

docs

Readthedocs documentation in reStructuredText format. The product of it's compilation is this documentation.

submodules

The nlu and memory submodules are imported at a specific commit from other repositories. Please do not change anything but the checked out commit within the Dialog System repository. In order to change nlu/memory functionality, refer to their specific repositories.

3.11.2 Where to extend functionalities

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. For more advanced functionality like *world interfaces* you'd need to implement different functionalities in different places. Refer the project structure above and the [Development tutorials](#) for this.

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

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.Conversation`, or create your own personality. Some static things, especially more complicated I/O devices or input analyzers need to be added to `robey.dialog.ConversationManager` or other places. Please have a look at [Development tutorials](#).

3.12 Project Constraints

3.12.1 Technical Constraints

Table 1: Operating System Constraints

Constraint Name	Description
Ubuntu => 16.04	Default Roboy OS
Java => 1.8.0	Reasonably recent and stable Java release. Sun.audio was removed with JDK9, thus it will not compile for any newer Java versions.

Table 2: Programming Constraints

Constraint Name	Description
IntelliJ IDEA	Difficulties with importing the project to NetBeans and Eclipse
rosjava	Due to using both Java and ros
Java 1.8.0	Reasonably recent and stable Java release

3.13 API

```
template <V>
```

```
interface AbstractValue
```

```
    Stores a single value.
```

```
    On update, the value is overwritten.
```

```
    Subclassed by robey.context.AbstractValueHistory< K, V >, robey.context.ObservableValue< V >,
    robey.context.Value< V >
```

Public Functions

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

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

template <K, V>

interface `AbstractValueHistory`

Maintains a map containing many values.

These values are accessible over the `getLastNValues` method, in addition to *AbstractValue* methods.

Public Functions

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

Return the n newest values in the history.

default int `roboy.context.AbstractValueHistory< K, V >.getMaxLimit()`

When value count reaches `maxLimit`, adding a new value deletes the oldest.

Override to change threshold.

int `roboy.context.AbstractValueHistory< K, V >.getNumberOfValuesSinceStart()`

Returns the total amount of `updateValue()` calls made on this history.

boolean `roboy.context.AbstractValueHistory< K, V >.contains(V value)`

Returns if object is present in this history.

boolean `roboy.context.AbstractValueHistory< K, V >.purgeHistory()`

Empties the current history.

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.EmotionAction*, *roboy.dialog.action.FaceAction*,
roboy.dialog.action.SpeechAction

class `ActiveInterlocutor` : **public** *roboy::context::Value*<Interlocutor>

The context value to hold an Interlocutor instance.

class `ActiveInterlocutorUpdater` : **public** *roboy::context::InternalUpdater*<*ActiveInterlocutor*, Interlocutor>

The interface for DM to replace the Interlocutor value held in the target *ActiveInterlocutor* instance.

Public Functions

`roboy.context.contextObjects.ActiveInterlocutorUpdater.ActiveInterlocutorUpdater(ActiveInterlocutor)`

class `Agedater`

Public Functions

HashMap<String, Integer> `roboy.util.Agedater.determineAge(String datestring)`

A helper function to determine the age based on the birthdate.

Java >= 8 specific: `java.time.LocalDate`, `java.time.Period`, `java.time.ZoneId`

Return timeSpans in form of HasMap - years:Integer, months:Integer and days:Integer

Parameters

- datestring:

Private Members

```
final Logger robey.util.Agedater.LOGGER = LogManager.getLogger()
```

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 *robey.linguistics.sentenceanalysis.AnswerAnalyzer*, *robey.linguistics.sentenceanalysis.DictionaryBasedSentenceTy*, *robey.linguistics.sentenceanalysis.EmotionAnalyzer*, *robey.linguistics.sentenceanalysis.IntentAnalyzer*, *robey.linguistics.sentenceanalysis.OntologyNERAnalyzer*, *robey.linguistics.sentenceanalysis.OpenNLPParser*, *robey.linguistics.sentenceanalysis.OpenNLPPPOSTagger*, *robey.linguistics.sentenceanalysis.Preprocessor*, *robey.linguistics.sentenceanalysis.ProfanityAnalyzer*, *robey.linguistics.sentenceanalysis.SemanticParserAnalyzer*, *robey.linguistics.sentenceanalysis.SentenceAnalyzer*, *robey.linguistics.sentenceanalysis.SimpleTokenizer*

Public Functions

```
Interpretation robey.linguistics.sentenceanalysis.Analyzer.analyze(Interpretation sent
```

class

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?”).

Public Functions

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

Private Static Attributes

```
final Logger robey.linguistics.sentenceanalysis.AnswerAnalyzer.logger = LogManager.getLogger
```

class AnswerAnalyzerTest

Public Functions

```
void robey.linguistics.sentenceanalysis.AnswerAnalyzerTest.testName()
```

```
void robey.linguistics.sentenceanalysis.AnswerAnalyzerTest.testOccupation()
```

```
void robey.linguistics.sentenceanalysis.AnswerAnalyzerTest.testOrigin()
```

```
void robey.linguistics.sentenceanalysis.AnswerAnalyzerTest.testHobby()  
void robey.linguistics.sentenceanalysis.AnswerAnalyzerTest.testMovie()
```

Private Functions

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

Private Static Attributes

```
final SimpleTokenizer robey.linguistics.sentenceanalysis.AnswerAnalyzerTest.tokenizer =  
final OpenNLPPPOSTagger robey.linguistics.sentenceanalysis.AnswerAnalyzerTest.pos = new O  
final OpenNLPPParser robey.linguistics.sentenceanalysis.AnswerAnalyzerTest.parser = new O  
final AnswerAnalyzer robey.linguistics.sentenceanalysis.AnswerAnalyzerTest.answer = new A  
  
class AudioDirection : public robey::context::ValueHistory<DirectionVector>  
    Can hold the history of DirectionVectors sent from Audio.  
  
class AudioDirectionUpdater : public robey::context::ROSTopicUpdater<DirectionVector, AudioDirection>  
    Pushes new values sent by the Audio ROS topic into the AudioDirection value history.
```

Public Functions

```
robey.context.contextObjects.AudioDirectionUpdater.AudioDirectionUpdater(AudioDirection)
```

Protected Functions

```
synchronized void robey.context.contextObjects.AudioDirectionUpdater.update()  
RosSubscribers robey.context.contextObjects.AudioDirectionUpdater.getTargetSubscriber()  
  
class  
    Using Bing to perform speech to text.  
    Requires internet connection.
```

Public Functions

```
robey.io.BingInput.BingInput(RosMainNode node)  
Input robey.io.BingInput.listen()
```

Private Members

```
RosMainNode robey.io.BingInput.rosMainNode  
  
class  
    Uses Bing for text to speech.  
    Requires internet connection.
```

Public Functions

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

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

class

Start a conversation with telegram.

Try to detect a greeting or robey names or some key word that initialize the conversation.

Public Functions

```
robey.dialog.states.botboy.BotBoyState.BotBoyState(String stateIdentifiers, StateParam
```

```
Output robey.dialog.states.botboy.BotBoyState.act()
```

```
Output robey.dialog.states.botboy.BotBoyState.react(Interpretation input)
```

```
State robey.dialog.states.botboy.BotBoyState.getNextState()
```

Private Members

```
final Logger robey.dialog.states.botboy.BotBoyState.logger = LogManager.getLogger()
```

```
final String robey.dialog.states.botboy.BotBoyState.TRANSITION_INITIALIZED = "initialized"
```

```
State robey.dialog.states.botboy.BotBoyState.next
```

class

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

Isn't implemented yet.

Public Functions

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

class

Cerevoice text to speech.

Public Functions

```
robey.io.CerevoiceOutput.CerevoiceOutput(RosMainNode node)
```

```
void robey.io.CerevoiceOutput.act(List< Action > actions)
```

```
void robey.io.CerevoiceOutput.say(String text)
```

Private Members

```
RosMainNode robey.io.CerevoiceOutput.rosMainNode
```

class Chatbot

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.dialog.Chatbot.main(String[] args)
```

Private Static Attributes

```
final Logger robey.dialog.Chatbot.logger = LogManager.getLogger()
```

class

Public Functions

```
robey.dialog.states.gameStates.ChooseGameState.ChooseGameState(String stateIdentifier,  
Output robey.dialog.states.gameStates.ChooseGameState.act()  
Output robey.dialog.states.gameStates.ChooseGameState.react(Interpretation input)  
State robey.dialog.states.gameStates.ChooseGameState.getNextState()
```

Private Functions

```
String robey.dialog.states.gameStates.ChooseGameState.inferGame(Interpretation input)
```

Private Members

```
final Logger robey.dialog.states.gameStates.ChooseGameState.LOGGER = LogManager.getLogger()  
String robey.dialog.states.gameStates.ChooseGameState.game = null  
String robey.dialog.states.gameStates.ChooseGameState.suggestedGame = null
```

Private Static Attributes

```
final String robey.dialog.states.gameStates.ChooseGameState.TRANSITION_CHOSE_SNAPCHAT =  
final String robey.dialog.states.gameStates.ChooseGameState.TRANSITION_CHOSE_20_Q = "chos  
final String robey.dialog.states.gameStates.ChooseGameState.TRANSITION_EXIT = "exitGame"  
final RandomList<String> robey.dialog.states.gameStates.ChooseGameState.EXISTING_GAMES
```

interface CleanUp

Devices that need extra cleaning operation on destruction implement this.

Subclassed by *robey.io.MultiInputDevice*, *robey.io.MultiOutputDevice*, *robey.io.TelegramInput*

Public Functions

```
void robey.io.CleanUp.cleanup()
```

class

Uses the command line as input device.

Public Functions

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

Protected Functions

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

Private Members

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

class

Uses the command line as output device.

Public Functions

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

class Concept

Public Functions

```
robey.linguistics.Concept.Concept(String id)
```

```
String robey.linguistics.Concept.getId()
```

```
String robey.linguistics.Concept.toString()
```

```
boolean robey.linguistics.Concept.equals(Object obj)
```

```
int robey.linguistics.Concept.hashCode()
```

Private Members

```
String robey.linguistics.Concept.id
```

class ConfigManager

Public Static Attributes

```
boolean robey.util.ConfigManager.ROS_ENABLED = false
String robey.util.ConfigManager.ROS_MASTER_IP = "127.0.0.1"
List<String> robey.util.ConfigManager.ROS_ACTIVE_PKGS = new ArrayList<>()
String robey.util.ConfigManager.ACTION_CLIENT_SCRIPT = "/home/robey/workspace/Robey/src/robey_dialog/robey_dialog_client.py"
boolean robey.util.ConfigManager.DEBUG = true
boolean robey.util.ConfigManager.DEMO_MODE = false
boolean robey.util.ConfigManager.INFINITE_REPETITION = false
String robey.util.ConfigManager.INPUT = "cmdline"
List<String> robey.util.ConfigManager.OUTPUTS = new ArrayList<>()
String robey.util.ConfigManager.UDP_HOST_ADDRESS = "127.0.0.1"
int robey.util.ConfigManager.UDP_IN_SOCKET = 55555
int robey.util.ConfigManager.UDP_OUT_SOCKET = 55556
DatagramSocket robey.util.ConfigManager.DATAGRAM_SOCKET
int robey.util.ConfigManager.PARSER_PORT = -1
String robey.util.ConfigManager.PERSONALITY_FILE = "resources/personalityFiles/tutorial/ToyStateMachine.j"
String robey.util.ConfigManager.IBM_TTS_USER = ""
String robey.util.ConfigManager.IBM_TTS_PASS = ""
boolean robey.util.ConfigManager.CONTEXT_GUI_ENABLED = false
String robey.util.ConfigManager.TELEGRAM_API_TOKENS_FILE = ""
```

Package Static Functions

```
robey.util.ConfigManager.[static initializer] ()
```

Private Static Functions

```
static void robey.util.ConfigManager.initializeConfig()
```

This function reads the YAML config file and initializes all fields.

It is called only once at the beginning

Private Static Attributes

```
String robey.util.ConfigManager.yamlConfigFile = "config.properties"
```

class Context

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

Takes care of correct initialization. For usage examples, check out ContextTest.java

Public Functions

roboy.context.Context.Context ()

Builds the class to instance maps.

void roboy.context.Context.initializeROS (RosMainNode ros)

Starts up the external updaters (which need a ROS main node).

Parameters

- `ros`:

Public Members

`new ValueInterface<>(new FaceCoordinates())`

`new ValueInterface<>(new ActiveInterlocutor())`

`new HistoryInterface<>(new DialogTopics())`

`new HistoryInterface<>(new DialogIntents())`

`new HistoryInterface<>(new AudioDirection())`

`new HistoryInterface<>(new ROSTest())`

`new HistoryInterface<>(new ValueHistory<Integer>())`

final DialogTopicsUpdater roboy.context.Context.DIALOG_TOPICS_UPDATER

final DialogIntentsUpdater roboy.context.Context.DIALOG_INTENTS_UPDATER

final ActiveInterlocutorUpdater roboy.context.Context.ACTIVE_INTERLOCUTOR_UPDATER

final OtherQuestionsUpdater roboy.context.Context.OTHER_QUESTIONS_UPDATER

Package Attributes

Logger roboy.context.Context.LOGGER = LogManager.getLogger()

Private Functions

void roboy.context.Context.addToGUI (Object... elements)

Private Members

final Object roboy.context.Context.initializationLock = new Object()

final ArrayList roboy.context.Context.guiValues = new ArrayList()

final ArrayList roboy.context.Context.guiHistories = new ArrayList()

boolean roboy.context.Context.rosInitialized = false

AudioDirectionUpdater roboy.context.Context.AUDIO_ANGLES_UPDATER

ROSTestUpdater roboy.context.Context.ROS_TEST_UPDATER

final FaceCoordinatesObserver roboy.context.Context.FACE_COORDINATES_OBSERVER

```
class ContextGUI
```

A simple GUI showing the values and histories in the *Context* with their content.

Public Static Functions

```
static void robey.context.ContextGUI.run(List< AbstractValue > values, List< Abstra
```

Private Functions

```
robey.context.ContextGUI.ContextGUI(List< AbstractValue > values, List< AbstractVal
```

```
void robey.context.ContextGUI.prepareGUI()
```

```
void robey.context.ContextGUI.startFrame()
```

```
void robey.context.ContextGUI.updateValues()
```

```
void robey.context.ContextGUI.updateHistories()
```

Private Members

```
JFrame robey.context.ContextGUI.mainFrame
```

```
TitledBorder robey.context.ContextGUI.valueBorder
```

```
JPanel robey.context.ContextGUI.valuePanel
```

```
Map<AbstractValue, JLabel> robey.context.ContextGUI.valueDisplays
```

```
Map<AbstractValueHistory, JScrollPane> robey.context.ContextGUI.historyDisplays
```

```
TitledBorder robey.context.ContextGUI.historyBorder
```

```
JPanel robey.context.ContextGUI.historyPanel
```

```
JPanel robey.context.ContextGUI.controlPanel
```

```
List<AbstractValue> robey.context.ContextGUI.values
```

```
List<AbstractValueHistory> robey.context.ContextGUI.histories
```

Private Static Attributes

```
int robey.context.ContextGUI.MAX_HISTORY_VALUES = 50
```

```
int robey.context.ContextGUI.FULL_WIDTH = 600
```

```
int robey.context.ContextGUI.FULL_HEIGHT = 600
```

```
int robey.context.ContextGUI.ATTR_WIDTH = 590
```

```
int robey.context.ContextGUI.ATTR_HEIGHT = 80
```

```
int robey.context.ContextGUI.HISTORY_HEIGHT = 300
```

```
String robey.context.ContextGUI.NO_VALUE = "<not initialized>"
```

```
class ContextIntegrationTest
```


Public Functions

```
void roboy.context.ContextIntegrationTest.initializeRosAndContext ()
```

```
void roboy.context.ContextIntegrationTest.checkROSTopicUpdating ()
```

This test tests if the context is properly integrated into the external (ROS) environment.

For this test to work, start the TEST_TOPIC subscriber defined under RosSubscribers via ROS_ACTIVE_PKGS: roboy_test in config.properties

Also, publishing needs to be done externally (e.g. manually via [rostopic pub -r 1 /roboy std_msgs/String "data: 'Test'"]).

Private Members

```
RosMainNode roboy.context.ContextIntegrationTest.ros
```

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

```
class ContextTest
```

Public Functions

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

```
void roboy.context.ContextTest.setAndGetDialogIntents ()
```

```
void roboy.context.ContextTest.testInterlocutor ()
```

```
void roboy.context.ContextTest.timestampedHistoryTest ()
```

```
void roboy.context.ContextTest.historyLimitExceededTest ()
```

```
void roboy.context.ContextTest.timestampedHistoryLimitExceededTest ()
```

```
void roboy.context.ContextTest.testObserver ()
```

```
void roboy.context.ContextTest.audioDirectionsTest ()
```

Package Attributes

```
Context roboy.context.ContextTest.context = new Context()
```

```
class Conversation : public Thread
```

A *Conversation* is in charge of leading conversation with an interlocutor.

Its behaviour is defined through the StateBasedPersonality. It communicates with the interlocutor via a Multi-InputDevice and a MultiOutputDevice. The List of analyzers is used to make the input string machine understandable.

Public Functions

```
roboy.dialog.Conversation.Conversation(StateBasedPersonality personality, Personality Personality)
```

- personality: *roboy.dialog.personality.StateBasedPersonality* object.
- personalityFile: File that the personality shall be initialized from.

- `multiIn`: Inputs for this conversation to act on.
- `multiOut`: Outputs for this conversation to act to.
- `analyzers`: All analyzers necessary for analyzing the inputs from `multiIn`. Please provide these in correct order.

```
void roboy.dialog.Conversation.run()
```

Package Functions

```
synchronized void roboy.dialog.Conversation.endConversation()
```

Ends conversation and resets state to initial.

Does not reset gathered information.

Private Members

```
final Logger roboy.dialog.Conversation.logger = LogManager.getLogger("Conversation" + this.getId())
```

```
final MultiInputDevice roboy.dialog.Conversation.multiIn
```

```
final MultiOutputDevice roboy.dialog.Conversation.multiOut
```

```
final List<Analyzer> roboy.dialog.Conversation.analyzers
```

```
final File roboy.dialog.Conversation.personalityFile
```

```
final StateBasedPersonality roboy.dialog.Conversation.personality
```

```
volatile boolean roboy.dialog.Conversation.isRunning = true
```

```
volatile boolean roboy.dialog.Conversation.paused = false
```

```
List<Action> roboy.dialog.Conversation.actions
```

```
class ConversationManager
```

Central managing node for `roboy_dialog`.

ConversationManager coordinates conversation dispatching, running and stopping, IO flows and everything else that needs a central contact. *ConversationManager* assumes that it runs on an actual roboy if `ROS_ENABLED` is true in `config.properties`.

Public Static Functions

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

```
static void roboy.dialog.ConversationManager.spawnConversation(String uuid)
```

Creates and spawns a conversation for a chatuser.

Parameters

- `uuid`: should consist of "servicename-[uuid]", if input allows only a single user, set to "local"

Exceptions

- `IOException`: If conversation could not created.

```
static void roboy.dialog.ConversationManager.stopConversation(String uuid)
```

Stops conversation thread for `uuid`.

Parameters

- `uuid`: should consist of “servicename-[uuid]”, if input allows only a single user, set to “local”

static Long robpy.dialog.ConversationManager.getConversationThreadID (String uuid)
 returns the threadID of the conversation with interlocutor uuid

Return null if thread does not exist, threadID otherwise

Parameters

- `uuid`: should consist of “servicename-[uuid]”, if input allows only a single user, set to “local”

Protected Static Functions

static void robpy.dialog.ConversationManager.deregisterConversation (Conversation conversation)
 Deregisters a conversation from the conversationmanager.

Should only be called from a conversation when it ends.

Parameters

- `conversation`: The conversation object to be deregistered

Private Static Functions

static Conversation robpy.dialog.ConversationManager.createConversation (RosMainNode rosMainNode)
 Creates and initializes a new conversation thread.

Does not start the thread.

Return *robpy.dialog.Conversation* object. Fully initialized, ready to launch.

Parameters

- `rosMainNode`: ROS node. Set null if ROS_ENABLED=false
- `analyzers`: All analyzers necessary for analyzing the inputs from multiIn. Please provide these in correct order.
- `inference`: Inference engine. The better, the smarter robpy gets.
- `memory`: Robpy memory access. Without, we cannot remember anything and conversations stay shallow.

Exceptions

- `IOException`: In case the IOdevices could not be correctly initialized.

static void robpy.dialog.ConversationManager.demoReadyCheck ()
 Pauses execution until enter is hit if DEMO_MODE is configured.

Very useful for fairs and other demonstrations where you’d want to explain something before demonstrating the dialog systems capabilities

Exceptions

- `IOException`:

static void robpy.dialog.ConversationManager.commandMode ()

Assume command mode: In *ConversationManager* thread wait for commands on cmdline and manage conversations according to them.

Private Static Attributes

```
final Logger roboy.dialog.ConversationManager.logger = LogManager.getLogger()
final HashMap<String, Conversation> roboy.dialog.ConversationManager.conversations = new
RosMainNode roboy.dialog.ConversationManager.rosMainNode
List<Analyzer> roboy.dialog.ConversationManager.analyzers
Neo4jMemoryInterface roboy.dialog.ConversationManager.memory
```

class CoordinateSet

A coordinate set data structure for the interlocutor face.

Public Functions

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

Package Attributes

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

class

This state asks the interlocutor whether Roboy should demonstrate one of his abilities.

The abilities include: shake hand, recognize objects, show emotion, move body

Every time this state is entered, Roboy picks one of the abilities (that haven't been demonstrated yet) and asks the interlocutor whether it should be demonstrated. If all abilities were already demonstrated, one is chosen at random. The ability is demonstrated only if the interlocutor said yes (or similar).

Control flow:

- act(): "Would you like to see me doing {ability}?"
- listen()
- react():
 - if answer = yes: demonstrate ability, say final remark, take 'abilityWasDemonstrated' transition
 - otherwise: skip ability, say another final remark, take 'abilityDemonstrationSkipped' transition

DemonstrateAbilitiesState interface: 1) Fallback is not required. 2) Outgoing transitions that have to be defined:

- abilityWasDemonstrated: following state if the ability was demonstrated
- abilityDemonstrationSkipped: following state if the ability demonstration was skipped 3) No parameters are used.

Public Functions

```
roboy.dialog.states.expoStates.DemonstrateAbilitiesState.DemonstrateAbilitiesState (String)
Output roboy.dialog.states.expoStates.DemonstrateAbilitiesState.act ()
Output roboy.dialog.states.expoStates.DemonstrateAbilitiesState.react (Interpretation interpretation)
State roboy.dialog.states.expoStates.DemonstrateAbilitiesState.getNextState ()
```

Protected Functions

```
Set<String> roboy.dialog.states.expoStates.DemonstrateAbilitiesState.getRequiredTransitions ()
```

Private Functions

```
void roboy.dialog.states.expoStates.DemonstrateAbilitiesState.resetAvailableAbilities ()
    Resets the list of available abilities so that it contains all of them.

RoboyAbility roboy.dialog.states.expoStates.DemonstrateAbilitiesState.selectRandomAbility ()
    Selects one of the abilities from the availableAbilities list at random and removes it from the list.

    If the list becomes empty this way, resets it to the initial state

    Return one of the available abilities
```

Private Members

```
final RandomList<RoboyAbility> roboy.dialog.states.expoStates.DemonstrateAbilitiesState.availableAbilities
RoboyAbility roboy.dialog.states.expoStates.DemonstrateAbilitiesState.activeAbility
State roboy.dialog.states.expoStates.DemonstrateAbilitiesState.nextState
final Logger roboy.dialog.states.expoStates.DemonstrateAbilitiesState.logger = LogManager.getLogger (this)
```

Private Static Attributes

```
final String roboy.dialog.states.expoStates.DemonstrateAbilitiesState.TRANS_ABILITY_DEMONSTRATE_SKILLS
final String roboy.dialog.states.expoStates.DemonstrateAbilitiesState.TRANS_ABILITY_SKILL
```

class

Roboy Demonstrate Skills State.

This state will:

- offer the interlocutor to ask a general question, mathematical problem
- retrieve the semantic parser result
- compose an answer
- fall back in case of failure OR
- offer a joke / an amusing fact
- in case of POSITIVE sentiment say those

ExpoIntroductionState interface: 1) Fallback is required. 2) Outgoing transitions that have to be defined, following state if the question was answered or the joke/fact were told:

- robey,
- abilities,
- newPerson. 3) No parameters are used.

Public Functions

```
robey.dialog.states.expoStates.DemonstrateSkillsState.DemonstrateSkillsState(String state)
Output robey.dialog.states.expoStates.DemonstrateSkillsState.act()
Output robey.dialog.states.expoStates.DemonstrateSkillsState.react(Interpretation input)
State robey.dialog.states.expoStates.DemonstrateSkillsState.getNextState()
```

Public Static Attributes

```
final String robey.dialog.states.expoStates.DemonstrateSkillsState.INTENTS_HISTORY_ID =
```

Protected Functions

```
Set<String> robey.dialog.states.expoStates.DemonstrateSkillsState.getRequiredTransitions()
```

Private Functions

```
RoboySkillIntent robey.dialog.states.expoStates.DemonstrateSkillsState.detectSkill(String input)
```

Private Members

```
final String [] robey.dialog.states.expoStates.DemonstrateSkillsState.TRANSITION_NAMES
final String [] robey.dialog.states.expoStates.DemonstrateSkillsState.INTENT_NAMES = TR
final Logger robey.dialog.states.expoStates.DemonstrateSkillsState.LOGGER = LogManager.getLogger()
State robey.dialog.states.expoStates.DemonstrateSkillsState.nextState
RoboySkillIntent robey.dialog.states.expoStates.DemonstrateSkillsState.skillIntent = null
class DetectedEntity
```

Public Functions

```
robey.linguistics.DetectedEntity.DetectedEntity(int tokenIndex, Entity entity)
Entity robey.linguistics.DetectedEntity.getEntity()
int robey.linguistics.DetectedEntity.getTokenIndex()
String robey.linguistics.DetectedEntity.toString()
```

```
boolean roboy.linguistics.DetectedEntity.equals(Object obj)
int roboy.linguistics.DetectedEntity.hashCode()
```

Private Members

```
Entity roboy.linguistics.DetectedEntity.entity
int roboy.linguistics.DetectedEntity.tokenIndex
class DialogIntents : public roboy::context::ValueHistory<IntentValue>
    Store the history of intents.
```

Public Functions

```
int roboy.context.contextObjects.DialogIntents.getMaxLimit()
boolean roboy.context.contextObjects.DialogIntents.isAttributePresent(String attribute)
class DialogIntentsUpdater : public roboy::context::InternalUpdater<DialogIntents, IntentValue>
    Update the history of intents.
```

Public Functions

```
roboy.context.contextObjects.DialogIntentsUpdater.DialogIntentsUpdater(DialogIntents t)
class DialogStateMachine
    State machine to manage dialog states.
```

State based personality is built on top of this class.

Main functionality of this class enables loading dialog state machines from personality files. There is also an option to save an existing state machine to file.

Personality files are JSON files that define a set of dialog states and transitions between them (see examples in [resources/personalityFiles/tutorial/](#)). Every state definition in the file has an identifier and specifies the implementation (class name) for the state. During parsing of the personality file this class will take the class name and create a Java State object using Java reflection.

Subclassed by *roboy.dialog.personality.StateBasedPersonality*

Public Functions

```
roboy.dialog.DialogStateMachine.DialogStateMachine(InferenceEngine inference, Context context)
    Create an empty DialogStateMachine.
```

Use `loadFromFile(...)` to load definitions from the personality file after creation. Alternatively you can create and add States to the machine manually from code.

Parameters

- `context`: reference to the context of the conversation this Statemachine belongs to (will be passed accessible to every newly created State object)
- `rosMainNode`: reference to the RosMainNode that will be passed to every newly created State object

- `memory`: reference to Memory that will be passed to every newly created State object

robpy.dialog.DialogStateMachine.DialogStateMachine(InferenceEngine inference, Context context)
Create an empty OFFLINE *DialogStateMachine* without a reference to the RosMainNode and Memory.

States will not be able to access the RosMainNode and Memory functionality. This constructor is mainly used for testing.

InferenceEngine robpy.dialog.DialogStateMachine.getInference()

Getter for the InferenceEngine to be accessible externally.

Let semblance of intelligence penetrate the mind of Roboy

Return the reference to the inference

Context robpy.dialog.DialogStateMachine.getContext()

State robpy.dialog.DialogStateMachine.getInitialState()

Returns the initial state for this state machine.

Return initial state for this state machine

void robpy.dialog.DialogStateMachine.setInitialState(State initial)

Set the initial state of this state machine.

The state will be automatically added to the machine if not already added. If active state was null, it will be set to the new initial state.

Parameters

- `initial`: initial state

void robpy.dialog.DialogStateMachine.setInitialState(String identifier)

Set the initial state of this state machine using state identifier.

If there is no state with specified identifier, you will get an error message and the initial state will be set to null.

Parameters

- `identifier`: identifier of the state that should become the initial state

State robpy.dialog.DialogStateMachine.getActiveState()

Returns the active state for this state machine.

Return active state for this state machine

void robpy.dialog.DialogStateMachine.setActiveState(State s)

Set the active state of this state machine.

The state will be automatically added to the machine if not already added.

Parameters

- `s`: state to make active

void robpy.dialog.DialogStateMachine.setActiveState(String identifier)

Set the active state using state identifier.

If there is no state with specified identifier, you will get an error message and active state will be set to null.

Parameters

- `identifier`: identifier of the state that should become the active state

State `roboy.dialog.DialogStateMachine.getStateByIdentifier(String identifier)`

Returns a state with given identifier.

Returns null if no such state was previously added to the machine.

Return state with given identifier or null

Parameters

- `identifier`: identifier of the state to retrieve

void `roboy.dialog.DialogStateMachine.addState(State s)`

Add a state to this state machine.

Parameters

- `s`: state to add.

void `roboy.dialog.DialogStateMachine.loadFromString(String s)`

Loads state machine from a JSON string.

The string must be a valid personality (usually loaded from a personality file).

Parameters

- `s`: personality string

void `roboy.dialog.DialogStateMachine.loadFromFile(File f)`

Loads state machine from a personality file.

File must contain a valid personality definition.

Parameters

- `f`: file with the personality definition

Exceptions

- `FileNotFoundException`: if file is not found

void `roboy.dialog.DialogStateMachine.saveToFile(File f)`

Save this state machine to a personality file in JSON format.

Parameters

- `f`: file to save

String `roboy.dialog.DialogStateMachine.toJsonString()`

Creates a JSON string that represents this state machine.

The JSON string is different from the `toString` representation which is more readable.

Return JSON string that represents this state machine

String `roboy.dialog.DialogStateMachine.toString()`

boolean `roboy.dialog.DialogStateMachine.equals(Object obj)`

Private Functions

void `roboy.dialog.DialogStateMachine.loadFromJSON(JsonElement json)`

Main function that parses a JSON personality object and creates a state machine.

Required properties:

- initialState (string identifier)
- states (array of state definitions) Optional properties:
- comment (personality file comment)

Parameters

- json: json object with the personality definition

StateParameters roboy.dialog.DialogStateMachine.parseStateParameters (JsonObject stateJ)
Parses parameters from the state json object.

A new instance of StateParameters is created.

Return StateParameters instance with all parameters defined in json object

Parameters

- stateJsO: json object representing a state

void roboy.dialog.DialogStateMachine.parseAndCreateStates (JsonArray statesJsA)
Parses every element of the json array and creates a state java object.

State parameters are parsed before the object is created.

Parameters

- statesJsA: json array containing states

void roboy.dialog.DialogStateMachine.parseAndSetTransitionsAndFallbacks (JsonArray stateJsA)
Parses every element of the json array (containing states).

For every state, finds and sets the fallback if defined. State transitions are also initialized.

Parameters

- statesJsA: json array containing states
- idToState: initialized hash map that resolves state IDs to state java objects

void roboy.dialog.DialogStateMachine.checkSuccessfulInitialization (HashMap< String, State> idToState)
For every state in the idToState hash map, check if all required transitions and parameters were initialized correctly.

Also check if all required fallbacks were set.

JsonObject roboy.dialog.DialogStateMachine.toJsonObject ()
Creates a JSON object that represents this state machine.

Return JSON object that represents this state machine

Private Members

final Logger roboy.dialog.DialogStateMachine.logger = LogManager.getLogger()

HashMap<String, State> roboy.dialog.DialogStateMachine.identifierToState
maps string identifiers to state objects ("Greeting" -> {GreetingState}) allows to have multiple instances of the same state class with different identifiers ("Greeting2" -> {GreetingState})

State roboy.dialog.DialogStateMachine.activeState

State roboy.dialog.DialogStateMachine.initialState

```
final RosMainNode robey.dialog.DialogStateMachine.rosMainNode
```

RosMainNode will be passed to every state as parameter.

```
final Neo4jMemoryInterface robey.dialog.DialogStateMachine.memory
```

```
final InferenceEngine robey.dialog.DialogStateMachine.inference
```

```
final Context robey.dialog.DialogStateMachine.context
```

```
HashMap<String, String> robey.dialog.DialogStateMachine.optionalPersFileInfo
```

Personality file additional information: everything like comment goes here.

[!!] Do not use it in your State implementation! This info is only stored to make sure we don't lose the comment etc. when saving this dialog state machine to file.

```
class DialogSystem
```

This class is deprecated.

It stays in the summer semester 2018 to ensure backward compatibility with some command line commands. Later it will be removed completely. Use *ConversationManager* instead. Please use ConversationManger instead.

Public Static Functions

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

```
class DialogTopics : public robey::context::ValueHistory<String>
```

A value history to hold the utterances from the interlocutor and Roboy.

```
class DialogTopicsUpdater : public robey::context::InternalUpdater<DialogTopics, String>
```

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

Public Functions

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

```
class
```

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.

Public Functions

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

Private Functions

```
SentenceType robey.linguistics.sentenceanalysis.DictionaryBasedSentenceTypeDetector.detect(Sentence sentence)
```

```
class DictionaryBasedSentenceTypeDetectorTest
```

Public Functions

```
void robey.linguistics.sentenceanalysis.DictionaryBasedSentenceTypeDetectorTest.testWhisper(Sentence sentence)
```

Private Members

```
DictionaryBasedSentenceTypeDetector roboy.linguistics.sentenceanalysis.DictionaryBasedSentenceTypeDetector
SimpleTokenizer roboy.linguistics.sentenceanalysis.DictionaryBasedSentenceTypeDetector
class DirVec : public DirectionVector
```

Public Functions

```
double roboy.context.ContextTest.DirVec.getAzimutalAngle()
void roboy.context.ContextTest.DirVec.setAzimutalAngle(double v)
double roboy.context.ContextTest.DirVec.getPolarAngle()
void roboy.context.ContextTest.DirVec.setPolarAngle(double v)
RawMessage roboy.context.ContextTest.DirVec.toRawMessage()
```

Package Attributes

```
double roboy.context.ContextTest.DirVec.azimutal_angle
double roboy.context.ContextTest.DirVec.polar_angle
class
```

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.

Public Functions

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

Package Attributes

```
DoubleMetaphone roboy.linguistics.phonetics.DoubleMetaphoneEncoder.doubleMetaphone
class
```

Public Functions

```
roboy.dialog.tutorials.tutorialStates.DoYouKnowMathState.DoYouKnowMathState(String state)
Output roboy.dialog.tutorials.tutorialStates.DoYouKnowMathState.act()
Output roboy.dialog.tutorials.tutorialStates.DoYouKnowMathState.react(Interpretation interpretation)
State roboy.dialog.tutorials.tutorialStates.DoYouKnowMathState.getNextState()
```

Private Members

State `roboy.dialog.tutorials.tutorialStates.DoYouKnowMathState.next`

class

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

Public Functions

roboy.memory.DummyMemory.DummyMemory ()

boolean roboy.memory.DummyMemory.save (MemoryNodeModel node)

This function is a dummy to use without ROS connection to *Neo4jMemory*.

Return true for success, false for fail

Parameters

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

String roboy.memory.DummyMemory.getId (int id)

This function is a dummy to use without ROS connection to *Neo4jMemory*.

Return Node representation of the result.

Parameters

- `id`: the ID of requested

ArrayList<Integer> roboy.memory.DummyMemory.getByQuery (MemoryNodeModel query)

This function is a dummy to use without ROS connection to *Neo4jMemory*.

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

Parameters

- `query`: the ID of requested

int roboy.memory.DummyMemory.create (MemoryNodeModel query)

boolean roboy.memory.DummyMemory.remove (MemoryNodeModel query)

This function is a dummy to use without ROS connection to *Neo4jMemory*.

Parameters

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

Private Static Attributes

final Logger roboy.memory.DummyMemory.logger = LogManager.getLogger()

class

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

Public Functions

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

Constructor.

Duration is set to 1.

Parameters

- **state**: The emotional expression. Possible values: angry, neutral, speak Please use RoboyEmotions instead of state Strings.

roboy.dialog.action.EmotionAction.EmotionAction(RoboyEmotion state)

Constructor.

Duration is set to 1.

Parameters

- **state**: The emotional expression. Please refer *roboy.emotions.RoboyEmotion* for supported emotions.

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

Constructor.

Parameters

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

roboy.dialog.action.EmotionAction.EmotionAction(RoboyEmotion state, int duration)

Constructor.

Duration is set to 1.

Parameters

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

String roboy.dialog.action.EmotionAction.getState()

int roboy.dialog.action.EmotionAction.getDuration()

Private Members

String roboy.dialog.action.EmotionAction.state

int roboy.dialog.action.EmotionAction.duration

class

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.

Public Functions

`Interpretation roboy.linguistics.sentenceanalysis.EmotionAnalyzer.analyze(Interpretation)`
class
 Roboy's facial expression output.

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)
roboy.linguistics.Entity.Entity (String key, String term)
String roboy.linguistics.Entity.getForm (String form)
String roboy.linguistics.Entity.getBaseForm ()
Map<String,String> roboy.linguistics.Entity.getForms ()
String roboy.linguistics.Entity.toString ()
boolean roboy.linguistics.Entity.equals (Object obj)
int roboy.linguistics.Entity.hashCode ()
```

Private Members

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

class
 Expo Introduction State.

This state will:

- ask the interlocutor for his name
- create and update the interlocutor in the context
- take one transition: roboyInfo

ExpoIntroductionState interface: 1) Fallback is not required. 2) Outgoing transitions that have to be defined, following state if Roboy introduced himself:

- skills,

- robey,
- abilities,
- newPerson. 3) Used 'infoFile' parameter containing Roboy answer phrases. Requires a path to RoboyInfoList.json

Public Functions

```
robey.dialog.states.expoStates.ExpoIntroductionState.ExpoIntroductionState(String state)
Output robey.dialog.states.expoStates.ExpoIntroductionState.act()
Output robey.dialog.states.expoStates.ExpoIntroductionState.react(Interpretation input)
State robey.dialog.states.expoStates.ExpoIntroductionState.getNextState()
```

Public Static Attributes

```
final String robey.dialog.states.expoStates.ExpoIntroductionState.INTENTS_HISTORY_ID = 'INTENTS_HISTORY_ID'
```

Private Functions

```
String robey.dialog.states.expoStates.ExpoIntroductionState.getNameFromInput(Interpretation input)
String robey.dialog.states.expoStates.ExpoIntroductionState.getRoboyFactsPhrase(RoboyFacts facts)
String robey.dialog.states.expoStates.ExpoIntroductionState.getIntroPhrase()
String robey.dialog.states.expoStates.ExpoIntroductionState.getResponsePhrase(String name)
State robey.dialog.states.expoStates.ExpoIntroductionState.getTransitionRandomly(String state)
void robey.dialog.states.expoStates.ExpoIntroductionState.updateInterlocutorInContext(Interlocutor interlocutor)
```

Private Members

```
final String robey.dialog.states.expoStates.ExpoIntroductionState.SELECTED_SKILLS = "skills"
final String robey.dialog.states.expoStates.ExpoIntroductionState.SELECTED_ABILITIES = "abilities"
final String robey.dialog.states.expoStates.ExpoIntroductionState.SELECTED_ROBOY_QA = "robey_qa"
final String robey.dialog.states.expoStates.ExpoIntroductionState.LEARN_ABOUT_PERSON = "learn_about_person"
final String robey.dialog.states.expoStates.ExpoIntroductionState.INFO_FILE_PARAMETER = "info_file_parameter"
final Logger robey.dialog.states.expoStates.ExpoIntroductionState.LOGGER = LogManager.getLogger(ExpoIntroductionState.class)

    "Oh wow.. Sorry for my confusion today. But what's your name?", "Mamma mia. So many people passing
    by today. Good you stopped by to talk to me. Could you tell me your name?", "Ehm, sorry.. Who am I
    currently talking to? These lights are so bright I cant see your face" ) ]

final RandomList<String> robey.dialog.states.expoStates.ExpoIntroductionState.responses
QAJsonParser robey.dialog.states.expoStates.ExpoIntroductionState.infoValues
State robey.dialog.states.expoStates.ExpoIntroductionState.nextState
```


class

Used for Hannover Messe 2018 states.

Extends the [State](#) class of the dialog state system. Expo dialog states that require probabilistic transitioning should extend this class. Uses uniform distribution.

Extends the [State](#) class with:

- `getTransitionRandomly` - chooses a named transition randomly from the transition names and intent names.
- `chooseIntentAttribute` - chooses the attribute of the intent randomly, so that it is not similar to the last ones.
- `lastNIntentsContainAttribute` - checks if the last N IntentValues of the IntentsHistory contain the given attribute.

Subclassed by *roboy.dialog.states.expoStates.DemonstrateSkillsState*, *roboy.dialog.states.expoStates.ExpoIntroductionState*, *roboy.dialog.states.expoStates.PersonalInformationAskingState*, *roboy.dialog.states.expoStates.RoboyQAState*

Public Functions

roboy.dialog.states.definitions.ExpoState.ExpoState(String stateIdentifier, StateParam

Create a state object with given identifier (state name) and parameters.

The parameters should contain a reference to a state machine for later use. The state will not automatically add itself to the state machine.

Parameters

- `stateIdentifier`: identifier (name) of this state
- `params`: parameters for this state, should contain a reference to a state machine

final State roboy.dialog.states.definitions.ExpoState.getTransitionRandomly(String[] t

Chooses a named transition randomly.

Gets transition names and intent names. Upon choosing the transition checks if the corresponding intent is a valid Neo4jProperty. If so, requests the attribute based on the property and saves property and attribute in the IntentsHistory. If the attribute is unrecoverable, returns the current state. Otherwise, returns the chosen state. When the dice hits a bigger number, returns the current state.

Return `transitionNames[i] -> State`

Parameters

- `transitionNames`:
- `intentNames`:
- `intentsHistoryId`:

Exceptions

- `IllegalArgumentException`:

Private Functions

String roboy.dialog.states.definitions.ExpoState.chooseIntentAttribute(Neo4jProperty p

Chooses the attribute of the intent randomly, so that it is not similar to the last n intents.

Return String containing the chosen value

Parameters

- predicate:
- evaluateLastN:

boolean roboy.dialog.states.definitions.ExpoState.lastNIntentsContainAttribute (String
Checks if the last N IntentValues of the IntentsHistory contain the given attribute.

Return true if contains otherwise false

Parameters

- attribute:
- n:

Private Members

final Logger roboy.dialog.states.definitions.ExpoState.LOGGER = LogManager.getLogger()

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.PeriodicUpdater< Target >*, *roboy.context.ROSTopicUpdater< Message, Target >*

Protected Functions

abstract void roboy.context.ExternalUpdater.update ()

class

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

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 (RoboyEmotion 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

roboy.dialog.action.FaceAction.FaceAction(RoboyEmotion state, int duration)
 Constructor.

Duration is set to 1.

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 : public *roboy::context::ObservableValue<CoordinateSet>*
 xzy-coordinates of a person in the field of vision.

class FaceCoordinatesObserver : public Observer
 Currently dummy functionality.

In the future, could trigger the rotation of the head towards the speaker. Observes the last location head was turned towards, and calls action if difference passes a threshold.

TODO: To implement head turning, change the dummy functionality in turnHead() method.

Public Functions

roboy.context.contextObjects.FaceCoordinatesObserver.FaceCoordinatesObserver()

void roboy.context.contextObjects.FaceCoordinatesObserver.update(Observable observable)

void roboy.context.contextObjects.FaceCoordinatesObserver.turnHead(double x, double y,

Package Attributes

double roboy.context.contextObjects.FaceCoordinatesObserver.lastUpdatedX

double roboy.context.contextObjects.FaceCoordinatesObserver.lastUpdatedY

double roboy.context.contextObjects.FaceCoordinatesObserver.lastUpdatedZ

long roboy.context.contextObjects.FaceCoordinatesObserver.nextUpdateTime

ExecutorService roboy.context.contextObjects.FaceCoordinatesObserver.executor

Package Static Attributes

```
double roboy.context.contextObjects.FaceCoordinatesObserver.TRIGGER_DIFFERENCE = 0.5
long roboy.context.contextObjects.FaceCoordinatesObserver.UPDATE_INTERVAL_MILLIS = 10000
```

class

This state ends the conversation.

FarewellState interface: 1) Fallback is not required. 2) This state has no outgoing transitions. 3) No parameters are used.

Public Functions

```
roboy.dialog.states.ordinaryStates.FarewellState.FarewellState(String stateIdentifier,
Output roboy.dialog.states.ordinaryStates.FarewellState.act()
Output roboy.dialog.states.ordinaryStates.FarewellState.react(Interpretation input)
State roboy.dialog.states.ordinaryStates.FarewellState.getNextState()
boolean roboy.dialog.states.ordinaryStates.FarewellState.isFallbackRequired()
```

Protected Functions

```
Set<String> roboy.dialog.states.ordinaryStates.FarewellState.getRequiredTransitionNames()
Set<String> roboy.dialog.states.ordinaryStates.FarewellState.getRequiredParameterNames()
```

Private Members

```
State roboy.dialog.states.ordinaryStates.FarewellState.next = null
int roboy.dialog.states.ordinaryStates.FarewellState.loops = 0
```

Private Static Attributes

```
final int roboy.dialog.states.ordinaryStates.FarewellState.MAX_LOOP_COUNT = 2
    "What a nice conversation! I have to think about everything we" + " were talking about. Let's talk again
    next time.", "I feel tired now, maybe my battery is low? Let's talk again later.", "Don't you think that the
    dialog team is amazing? They are happy to " + "tell you more about my system. Just ask one of them!") ]
```

class FileLineReader

Public Static Functions

```
static RandomList<String> roboy.util.FileLineReader.readFile(String path)
```

class

Free TTS text to speech.

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
```

Public Functions

```
roboy.dialog.states.gameStates.GamingSnapchatState.GamingSnapchatState (String stateId)
Output roboy.dialog.states.gameStates.GamingSnapchatState.act ()
Output roboy.dialog.states.gameStates.GamingSnapchatState.react (Interpretation input)
State roboy.dialog.states.gameStates.GamingSnapchatState.getNextState ()
```

Private Functions

```
boolean roboy.dialog.states.gameStates.GamingSnapchatState.checkUserSaidStop (Interpretation input)
Map<String, List<String> > roboy.dialog.states.gameStates.GamingSnapchatState.buildSynonyms ()
```

Private Members

```
Map<String, List<String> > roboy.dialog.states.gameStates.GamingSnapchatState.EXISTING_FILTERS
RandomList<String> roboy.dialog.states.gameStates.GamingSnapchatState.filters
final Logger roboy.dialog.states.gameStates.GamingSnapchatState.LOGGER = LogManager.getLogger (GamingSnapchatState.class)
final Inference roboy.dialog.states.gameStates.GamingSnapchatState.localInference = new Inference ()
boolean roboy.dialog.states.gameStates.GamingSnapchatState.filterApplied = false
List<String> roboy.dialog.states.gameStates.GamingSnapchatState.desiredFilters = new ArrayList<> ()
boolean roboy.dialog.states.gameStates.GamingSnapchatState.stopGame = false
String roboy.dialog.states.gameStates.GamingSnapchatState.suggestedFilter = ""
```

Private Static Attributes

```
final String roboy.dialog.states.gameStates.GamingSnapchatState.TRANSITION_GAME_ENDED =  
final String roboy.dialog.states.gameStates.GamingSnapchatState.EXISTING_FILTERS_ID = "f"  
class
```

Public Functions

```
roboy.dialog.states.gameStates.GamingTwentyQuestionsState.GamingTwentyQuestionsState (S  
Output roboy.dialog.states.gameStates.GamingTwentyQuestionsState.act ()  
Output roboy.dialog.states.gameStates.GamingTwentyQuestionsState.react (Interpretation  
State roboy.dialog.states.gameStates.GamingTwentyQuestionsState.getNextState ()
```

Private Functions

```
String roboy.dialog.states.gameStates.GamingTwentyQuestionsState.getIntent (Interpretat  
Output roboy.dialog.states.gameStates.GamingTwentyQuestionsState.askNextQuestion ()  
Output roboy.dialog.states.gameStates.GamingTwentyQuestionsState.saveUsersEstimate (Str  
Output roboy.dialog.states.gameStates.GamingTwentyQuestionsState.doAGuess ()  
Output roboy.dialog.states.gameStates.GamingTwentyQuestionsState.processUserGuessAnswer  
void roboy.dialog.states.gameStates.GamingTwentyQuestionsState.resetGame ()  
boolean roboy.dialog.states.gameStates.GamingTwentyQuestionsState.checkUserSaidStop (In  
void roboy.dialog.states.gameStates.GamingTwentyQuestionsState.applyFilter (String winn
```

Private Members

```
final Logger roboy.dialog.states.gameStates.GamingTwentyQuestionsState.LOGGER = LogManag  
Akiwrapper roboy.dialog.states.gameStates.GamingTwentyQuestionsState.aw = new AkiwrapperBui  
Question roboy.dialog.states.gameStates.GamingTwentyQuestionsState.nextQuestion = null  
Guess roboy.dialog.states.gameStates.GamingTwentyQuestionsState.currentGuess = null  
List<Long> roboy.dialog.states.gameStates.GamingTwentyQuestionsState.declined = new ArrayL  
boolean roboy.dialog.states.gameStates.GamingTwentyQuestionsState.userReady = false  
boolean roboy.dialog.states.gameStates.GamingTwentyQuestionsState.guessesAvailable = fals  
boolean roboy.dialog.states.gameStates.GamingTwentyQuestionsState.gameFinished = false  
boolean roboy.dialog.states.gameStates.GamingTwentyQuestionsState.stopGame = false  
boolean roboy.dialog.states.gameStates.GamingTwentyQuestionsState.filterApplied = false  
boolean roboy.dialog.states.gameStates.GamingTwentyQuestionsState.emotionShown = false  
String roboy.dialog.states.gameStates.GamingTwentyQuestionsState.winner = ""
```

Private Static Attributes

```
final double roboy.dialog.states.gameStates.GamingTwentyQuestionsState.PROBABILITY_THR
```

```
final String roboy.dialog.states.gameStates.GamingTwentyQuestionsState.TRANSITION_GAME
```

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

```
class HistoryInterface
```

This is the interface over which *Context* value histories can be queried.

Initialize as static field of the *Context* class. Add your *ValueHistory* implementation class, its key and return types as generic parameters.

Parameters

- <I>: An implementation of *AbstractValueHistory*.
- <K>: The keys used within the History instance.
- <V>: The type of data stored within the History instance.

Public Functions

```
Map<K, V> roboy.context.HistoryInterface< I extends AbstractValueHistory< K, V, K, V >
```

Get n elements saved into the corresponding *ValueHistory* instance (or all elements, if all < n).

```
V roboy.context.HistoryInterface< I extends AbstractValueHistory< K, V, K, V >.getLast
```

Get the last element saved into the corresponding *ValueHistory* instance.

```
int roboy.context.HistoryInterface< I extends AbstractValueHistory< K, V, K, V >.value
```

Get the total nr of times a new value was saved into the corresponding *ValueHistory* instance.

Note: as histories can be limited in size, less elements might be actually stored than the total.

```
boolean roboy.context.HistoryInterface< I extends AbstractValueHistory< K, V, K, V >.c
```

Check if the object exists among the valueHistory values.

```
boolean roboy.context.HistoryInterface< I extends AbstractValueHistory< K, V, K, V >.p
```

Removes all the valueHistory values.

Protected Functions

```
roboy.context.HistoryInterface< I extends AbstractValueHistory< K, V, K, V >.HistoryIn
```

Protected Attributes

```
I roboy.context.HistoryInterface< I extends AbstractValueHistory< K, V, K, V >.valueHi
```

Package Functions

```
I roboy.context.HistoryInterface< I extends AbstractValueHistory< K, V, K, V >.getCont
```

```
class
```

Uses IBM Cloud text to speech.

Requires internet connection and valid IBM Bluemix credentials.

Public Functions

```
roboy.io.IBMWatsonOutput.IBMWatsonOutput ()  
void roboy.io.IBMWatsonOutput.act (List< Action > actions)  
void roboy.io.IBMWatsonOutput.say (String text)
```

Package Attributes

```
final Logger roboy.io.IBMWatsonOutput.logger = LogManager.getLogger()
```

Private Members

```
TextToSpeech roboy.io.IBMWatsonOutput.synthesizer  
class
```

Public Functions

HashMap<Neo4jProperty, String> roboy.logic.Inference.inferProperties (ArrayList< Neo4jProperty> keys)
Basic inference method Infers the property information with regard to the requested keys, tries to extract and ground the information from the available Interpretation.

Return HashMap containing properties and inferred data/null if NA

Parameters

- keys:

String roboy.logic.Inference.inferProperty (Neo4jProperty key, Interpretation input)
Basic inference method Infers the property information with regard to the requested keys, tries to extract and ground the information from the available Interpretation.

Return String containing inferred result/null if NA

Parameters

- key:

HashMap<Neo4jRelationship, String> roboy.logic.Inference.inferRelationships (ArrayList< Neo4jRelationship> keys)
Basic inference method Infers the relationship information with regard to the requested keys, tries to extract and ground the information from the available Interpretation.

Return HashMap containing relationships and inferred data/null if NA

Parameters

- keys:

String roboy.logic.Inference.inferRelationship (Neo4jRelationship key, Interpretation input)
Basic inference method Infers the relationship information with regard to the requested keys, tries to extract and ground the information from the available Interpretation.

Return String containing inferred result/null if NA

Parameters

- key:

HashMap<Neo4jLabel, String> roboy.logic.Inference.inferLabels(ArrayList< Neo4jLabel :
Basic inference method Infers the label information with regard to the requested keys, tries to extract and ground the information from the available Interpretation.

Return HashMap containing labels and inferred data/null if NA

Parameters

- keys:

String roboy.logic.Inference.inferLabel(Neo4jLabel key, Interpretation input)
Basic inference method Infers the label information with regard to the requested keys, tries to extract and ground the information from the available Interpretation.

Return String containing inferred result/null if NA

Parameters

- key:

Linguistics.UtteranceSentiment roboy.logic.Inference.inferSentiment(Interpretation inp
Basic inference method Infers the sentiment of the utterance tries to extract and ground the information from the available Interpretation.

Return String containing inferred result/null if NA

Parameters

- input:

List<String> roboy.logic.Inference.inferSnapchatFilter(Interpretation input, Map< Stri

Package Attributes

final Logger roboy.logic.Inference.LOGGER = LogManager.getLogger()

Package Static Attributes

final List<String> roboy.logic.Inference.positiveTokens = Arrays.asList("yes", "yep", "yeah", "ok", "s

final List<String> roboy.logic.Inference.negativeTokens = Arrays.asList("no", "nope", "later", "not", "

final List<String> roboy.logic.Inference.uncertaintyTokens = Arrays.asList("guess", "probably", "cou

Private Functions

String roboy.logic.Inference.inferName(Interpretation input)

interface InferenceEngine

Subclassed by *roboy.logic.Inference*

Public Functions

HashMap<Neo4jProperty, String> roboy.logic.InferenceEngine.inferProperties (ArrayList<

Basic inference method Infers the property information with regard to the requested keys, tries to extract and ground the information from the available Interpretation.

Return HashMap containing properties and inferred data/null if NA

Parameters

- keys:

String roboy.logic.InferenceEngine.inferProperty (Neo4jProperty key, Interpretation inp

Basic inference method Infers the property information with regard to the requested keys, tries to extract and ground the information from the available Interpretation.

Return String containing inferred result/null if NA

Parameters

- key:

HashMap<Neo4jRelationship, String> roboy.logic.InferenceEngine.inferRelationships (Arra

Basic inference method Infers the relationship information with regard to the requested keys, tries to extract and ground the information from the available Interpretation.

Return HashMap containing relationships and inferred data/null if NA

Parameters

- keys:

String roboy.logic.InferenceEngine.inferRelationship (Neo4jRelationship key, Interpreta

Basic inference method Infers the relationship information with regard to the requested keys, tries to extract and ground the information from the available Interpretation.

Return String containing inferred result/null if NA

Parameters

- key:

HashMap<Neo4jLabel, String> roboy.logic.InferenceEngine.inferLabels (ArrayList< Neo4jL

Basic inference method Infers the label information with regard to the requested keys, tries to extract and ground the information from the available Interpretation.

Return HashMap containing labels and inferred data/null if NA

Parameters

- keys:

String roboy.logic.InferenceEngine.inferLabel (Neo4jLabel key, Interpretation input)

Basic inference method Infers the label information with regard to the requested keys, tries to extract and ground the information from the available Interpretation.

Return String containing inferred result/null if NA

Parameters

- key:

UtteranceSentiment `roboy.logic.InferenceEngine.inferSentiment(Interpretation input)`
 Basic inference method Infers the sentiment of the utterance tries to extract and ground the information from the available Interpretation.

Return String containing inferred result/null if NA

Parameters

- `input`:

class `InferenceEngineTest`

Public Functions

`void roboy.logic.InferenceEngineTest.test()`

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, Interpretation attributes)`

`String roboy.io.Input.getSentence()`

`Interpretation roboy.io.Input.getAttributes()`

`void roboy.io.Input.setAttributes(Interpretation interpretation)`

Private Members

`String roboy.io.Input.sentence`

`Interpretation 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.TelegramInput*, *roboy.io.UdpInput*

Public Functions

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

class

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.

Public Functions

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

Private Members

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

class IntentValue

The value of the question intent based on Neo4j Relationship or a string.

Referenced by the intents history id to distinguish between the States which pushed the values to the history.

Public Functions

```
roboy.context.contextObjects.IntentValue.IntentValue(String intentsHistoryId, Neo4jRelationship rel)
roboy.context.contextObjects.IntentValue.IntentValue(String intentsHistoryId, Neo4jProperty prop)
roboy.context.contextObjects.IntentValue.IntentValue(String intentsHistoryId, String id)
roboy.context.contextObjects.IntentValue.IntentValue(String intentsHistoryId, Neo4jRelationship rel)
roboy.context.contextObjects.IntentValue.IntentValue(String intentsHistoryId, Neo4jProperty prop)
roboy.context.contextObjects.IntentValue.IntentValue(String intentsHistoryId, String id)
String roboy.context.contextObjects.IntentValue.getId()
Neo4jRelationship roboy.context.contextObjects.IntentValue.getNeo4jRelationshipValue()
Neo4jProperty roboy.context.contextObjects.IntentValue.getNeo4jPropertyValue()
String roboy.context.contextObjects.IntentValue.getStringValue()
String roboy.context.contextObjects.IntentValue.getAttribute()
boolean roboy.context.contextObjects.IntentValue.equals(Object obj)
int roboy.context.contextObjects.IntentValue.hashCode()
String roboy.context.contextObjects.IntentValue.toString()
```

Private Members

```
String roboy.context.contextObjects.IntentValue.id
Neo4jRelationship roboy.context.contextObjects.IntentValue.neo4jRelationshipValue = null
Neo4jProperty roboy.context.contextObjects.IntentValue.neo4jPropertyValue = null
String roboy.context.contextObjects.IntentValue.stringValue
String roboy.context.contextObjects.IntentValue.attribute = null
```

class

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 (Neo4jMemoryInterface memory)`

`roboy.memory.nodes.Interlocutor.Interlocutor (Neo4jMemoryInterface memory, String name)`

`roboy.memory.nodes.Interlocutor.Interlocutor (Neo4jMemoryInterface memory, Uuid uuid)`

`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.

`void roboy.memory.nodes.Interlocutor.addUuid (Uuid uuid)`

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

`Uuid roboy.memory.nodes.Interlocutor.getUuid (UuidType type)`

`boolean roboy.memory.nodes.Interlocutor.hasRelationship (Neo4jRelationship relationship)`

`ArrayList<Integer> roboy.memory.nodes.Interlocutor.getRelationships (Neo4jRelationship relationship)`

`void roboy.memory.nodes.Interlocutor.saveUzupisProperty (UzupisIntents intent, String value)`

`HashMap<UzupisIntents, String> roboy.memory.nodes.Interlocutor.getUzupisInfo ()`

`void roboy.memory.nodes.Interlocutor.addInformation (Neo4jRelationship relationship, String information)`

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

`RelationshipAvailability roboy.memory.nodes.Interlocutor.checkRelationshipAvailability (ArrayList<String> predicates)`

Checks if predicates from the input array are available for this interlocutor.

Return one of three: all, some or none available

Parameters

- `rels`: array of predicates to check

`HashMap<Boolean, ArrayList<Neo4jRelationship> > roboy.memory.nodes.Interlocutor.getPurposes ()`

Public Members

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

Private Functions

`boolean roboy.memory.nodes.Interlocutor.init (Interlocutor interlocutor)`

`MemoryNodeModel roboy.memory.nodes.Interlocutor.queryForMatchingNodes (Interlocutor interlocutor, String query)`

`MemoryNodeModel roboy.memory.nodes.Interlocutor.create (Interlocutor interlocutor, String name)`

`void roboy.memory.nodes.Interlocutor.set (MemoryNodeModel node)`

Private Members

```
HashMap<UzupisIntents, String> roboy.memory.nodes.Interlocutor.uzupisInfo = new HashMap<>()
```

```
final Logger roboy.memory.nodes.Interlocutor.LOGGER = LogManager.getLogger()
```

```
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.ActiveInterlocutorUpdater*, *roboy.context.contextObjects.DialogIntentsUpdater*, *roboy.context.contextObjects.DialogTopicsUpdater*, *roboy.context.contextObjects.OtherQuestionsUpdater*

Public Functions

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

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

Package Attributes

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

```
class Interpretation : public Cloneable
```

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()
```

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

```
roboy.linguistics.sentenceanalysis.Interpretation.Interpretation(String sentence, Inte
```

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

```
roboy.linguistics.sentenceanalysis.Interpretation.Interpretation(Interpretation interp
```

```
SentenceType roboy.linguistics.sentenceanalysis.Interpretation.getSentenceType()
```

```
void roboy.linguistics.sentenceanalysis.Interpretation.setSentenceType(SentenceType se
```

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

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

```
List<Triple> roboy.linguistics.sentenceanalysis.Interpretation.getTriples()
```

```
void roboy.linguistics.sentenceanalysis.Interpretation.setTriples(List< Triple > tri
```

```

List<Triple> roboy.linguistics.sentenceanalysis.Interpretation.getSemTriples ()
void roboy.linguistics.sentenceanalysis.Interpretation.setSemTriples(List< Triple > )
List<String> roboy.linguistics.sentenceanalysis.Interpretation.getTokens ()
void roboy.linguistics.sentenceanalysis.Interpretation.setTokens(List< String > tokens)
String [] roboy.linguistics.sentenceanalysis.Interpretation.getPosTags ()
void roboy.linguistics.sentenceanalysis.Interpretation.setPosTags(String[] posTags)
String [] roboy.linguistics.sentenceanalysis.Interpretation.getLemmas ()
void roboy.linguistics.sentenceanalysis.Interpretation.setLemmas(String[] lemmas)
List<DetectedEntity> roboy.linguistics.sentenceanalysis.Interpretation.getKeywords ()
void roboy.linguistics.sentenceanalysis.Interpretation.setKeywords(List< DetectedEnti
void roboy.linguistics.sentenceanalysis.Interpretation.addKeyword(DetectedEntity keywo
String roboy.linguistics.sentenceanalysis.Interpretation.getAssociation ()
void roboy.linguistics.sentenceanalysis.Interpretation.setAssociation(String associati
Map<SemanticRole, String> roboy.linguistics.sentenceanalysis.Interpretation.getPas ()
void roboy.linguistics.sentenceanalysis.Interpretation.setPas (Map< SemanticRole , Str
String roboy.linguistics.sentenceanalysis.Interpretation.getName ()
void roboy.linguistics.sentenceanalysis.Interpretation.setName (String name)
String roboy.linguistics.sentenceanalysis.Interpretation.getCelebrity ()
void roboy.linguistics.sentenceanalysis.Interpretation.setCelebrity (String celebrity)
boolean roboy.linguistics.sentenceanalysis.Interpretation.isRoboy ()
void roboy.linguistics.sentenceanalysis.Interpretation.setRoboy (boolean roboy)
String roboy.linguistics.sentenceanalysis.Interpretation.getObjAnswer ()
void roboy.linguistics.sentenceanalysis.Interpretation.setObjAnswer (String objAnswer)
String roboy.linguistics.sentenceanalysis.Interpretation.getPredAnswer ()
void roboy.linguistics.sentenceanalysis.Interpretation.setPredAnswer (String predAnswer)
RoboyEmotion roboy.linguistics.sentenceanalysis.Interpretation.getEmotion ()
void roboy.linguistics.sentenceanalysis.Interpretation.setEmotion (RoboyEmotion emotion)
boolean roboy.linguistics.sentenceanalysis.Interpretation.getProfanity ()
void roboy.linguistics.sentenceanalysis.Interpretation.setProfanity (boolean profanity)
String roboy.linguistics.sentenceanalysis.Interpretation.getIntent ()
void roboy.linguistics.sentenceanalysis.Interpretation.setIntent (String intent)
String roboy.linguistics.sentenceanalysis.Interpretation.getIntentDistance ()
void roboy.linguistics.sentenceanalysis.Interpretation.setIntentDistance (String intent)
String roboy.linguistics.sentenceanalysis.Interpretation.getParse ()
void roboy.linguistics.sentenceanalysis.Interpretation.setParse (String parse)
String roboy.linguistics.sentenceanalysis.Interpretation.getParseAnswer ()

```

```
void roboy.linguistics.sentenceanalysis.Interpretation.setParseAnswer(String parseAnswer)
String roboy.linguistics.sentenceanalysis.Interpretation.getUnderspecifiedTermQuestion()
void roboy.linguistics.sentenceanalysis.Interpretation.setUnderspecifiedTermQuestion(String termQuestion)
String roboy.linguistics.sentenceanalysis.Interpretation.getUnderspecifiedQuestion()
void roboy.linguistics.sentenceanalysis.Interpretation.setUnderspecifiedQuestion(String question)
String roboy.linguistics.sentenceanalysis.Interpretation.getUnderspecifiedAnswer()
void roboy.linguistics.sentenceanalysis.Interpretation.setUnderspecifiedAnswer(String answer)
UtteranceSentiment roboy.linguistics.sentenceanalysis.Interpretation.getSentiment()
void roboy.linguistics.sentenceanalysis.Interpretation.setSentiment(UtteranceSentiment sentiment)
String roboy.linguistics.sentenceanalysis.Interpretation.getUtteranceType()
void roboy.linguistics.sentenceanalysis.Interpretation.setUtteranceType(String utteranceType)
ParsingOutcome roboy.linguistics.sentenceanalysis.Interpretation.getParsingOutcome()
void roboy.linguistics.sentenceanalysis.Interpretation.setParsingOutcome(ParsingOutcome parsingOutcome)
String roboy.linguistics.sentenceanalysis.Interpretation.getAnswer()
void roboy.linguistics.sentenceanalysis.Interpretation.setAnswer(String answer)
void roboy.linguistics.sentenceanalysis.Interpretation.addTriple(Triple triple)
void roboy.linguistics.sentenceanalysis.Interpretation.copy(Interpretation interpretation)
void roboy.linguistics.sentenceanalysis.Interpretation.put(Interpretation interpretation)
String roboy.linguistics.sentenceanalysis.Interpretation.toString()
boolean roboy.linguistics.sentenceanalysis.Interpretation.equals(Object obj)
int roboy.linguistics.sentenceanalysis.Interpretation.hashCode()
```

Private Members

```
final Logger roboy.linguistics.sentenceanalysis.Interpretation.LOGGER = LogManager.getLogger(Interpretation.class)
SentenceType roboy.linguistics.sentenceanalysis.Interpretation.sentenceType = null
String roboy.linguistics.sentenceanalysis.Interpretation.sentence = null
List<Triple> roboy.linguistics.sentenceanalysis.Interpretation.triples = null
List<Triple> roboy.linguistics.sentenceanalysis.Interpretation.semTriples = null
List<String> roboy.linguistics.sentenceanalysis.Interpretation.tokens = null
String [] roboy.linguistics.sentenceanalysis.Interpretation.postTags = null
String [] roboy.linguistics.sentenceanalysis.Interpretation.lemmas = null
List<DetectedEntity> roboy.linguistics.sentenceanalysis.Interpretation.keywords = null
String roboy.linguistics.sentenceanalysis.Interpretation.association = null
Map<SemanticRole, String> roboy.linguistics.sentenceanalysis.Interpretation.pas = null
String roboy.linguistics.sentenceanalysis.Interpretation.name = null
String roboy.linguistics.sentenceanalysis.Interpretation.celebrity = null
```



```

boolean roboy.linguistics.sentenceanalysis.Interpretation.isRoboy = false
String roboy.linguistics.sentenceanalysis.Interpretation.objAnswer = null
String roboy.linguistics.sentenceanalysis.Interpretation.predAnswer = null
RoboyEmotion roboy.linguistics.sentenceanalysis.Interpretation.emotion = null
boolean roboy.linguistics.sentenceanalysis.Interpretation.profanity = false
String roboy.linguistics.sentenceanalysis.Interpretation.intent = null
String roboy.linguistics.sentenceanalysis.Interpretation.intentDistance = null
String roboy.linguistics.sentenceanalysis.Interpretation.parse = null
String roboy.linguistics.sentenceanalysis.Interpretation.parseAnswer = null
String roboy.linguistics.sentenceanalysis.Interpretation.underspecifiedTermQuestion = null
String roboy.linguistics.sentenceanalysis.Interpretation.underspecifiedQuestion = null
String roboy.linguistics.sentenceanalysis.Interpretation.underspecifiedAnswer = null
UtteranceSentiment roboy.linguistics.sentenceanalysis.Interpretation.sentiment = null
String roboy.linguistics.sentenceanalysis.Interpretation.utteranceType = null
ParsingOutcome roboy.linguistics.sentenceanalysis.Interpretation.parsingOutcome = null
String roboy.linguistics.sentenceanalysis.Interpretation.answer

class InterpretationTest

```

Public Functions

```

void roboy.linguistics.sentenceanalysis.InterpretationTest.setUp ()
void roboy.linguistics.sentenceanalysis.InterpretationTest.getSentenceType ()
void roboy.linguistics.sentenceanalysis.InterpretationTest.getSentence ()
void roboy.linguistics.sentenceanalysis.InterpretationTest.getTriples ()
void roboy.linguistics.sentenceanalysis.InterpretationTest.getSemTriples ()
void roboy.linguistics.sentenceanalysis.InterpretationTest.getTokens ()
void roboy.linguistics.sentenceanalysis.InterpretationTest.getPosTags ()
void roboy.linguistics.sentenceanalysis.InterpretationTest.getLemmas ()
void roboy.linguistics.sentenceanalysis.InterpretationTest.getKeywords ()
void roboy.linguistics.sentenceanalysis.InterpretationTest.getAssociation ()
void roboy.linguistics.sentenceanalysis.InterpretationTest.getPas ()
void roboy.linguistics.sentenceanalysis.InterpretationTest.getName ()
void roboy.linguistics.sentenceanalysis.InterpretationTest.getCelebrity ()
void roboy.linguistics.sentenceanalysis.InterpretationTest.isRoboy ()
void roboy.linguistics.sentenceanalysis.InterpretationTest.getObjAnswer ()
void roboy.linguistics.sentenceanalysis.InterpretationTest.getPredAnswer ()
void roboy.linguistics.sentenceanalysis.InterpretationTest.getEmotion ()

```

```
void robey.linguistics.sentenceanalysis.InterpretationTest.getIntent ()
void robey.linguistics.sentenceanalysis.InterpretationTest.getIntentDistance ()
void robey.linguistics.sentenceanalysis.InterpretationTest.getParse ()
void robey.linguistics.sentenceanalysis.InterpretationTest.getParseAnswer ()
void robey.linguistics.sentenceanalysis.InterpretationTest.getUnderspecifiedTermQuestion ()
void robey.linguistics.sentenceanalysis.InterpretationTest.getUnderspecifiedQuestion ()
void robey.linguistics.sentenceanalysis.InterpretationTest.getUnderspecifiedAnswer ()
void robey.linguistics.sentenceanalysis.InterpretationTest.getSentiment ()
void robey.linguistics.sentenceanalysis.InterpretationTest.getUtteranceType ()
void robey.linguistics.sentenceanalysis.InterpretationTest.getParsingOutcome ()
void robey.linguistics.sentenceanalysis.InterpretationTest.getAnswer ()
void robey.linguistics.sentenceanalysis.InterpretationTest.copy ()
```

Private Members

```
Interpretation robey.linguistics.sentenceanalysis.InterpretationTest.analyzedInterpretation
Interpretation robey.linguistics.sentenceanalysis.InterpretationTest.defaultInterpretation
Linguistics.SentenceType robey.linguistics.sentenceanalysis.InterpretationTest.sentenceType
String robey.linguistics.sentenceanalysis.InterpretationTest.sentence
Triple robey.linguistics.sentenceanalysis.InterpretationTest.triple
List<Triple> robey.linguistics.sentenceanalysis.InterpretationTest.triples
List<Triple> robey.linguistics.sentenceanalysis.InterpretationTest.semTriples
List<String> robey.linguistics.sentenceanalysis.InterpretationTest.tokens
String [] robey.linguistics.sentenceanalysis.InterpretationTest.postTags
String [] robey.linguistics.sentenceanalysis.InterpretationTest.lemmas
Entity robey.linguistics.sentenceanalysis.InterpretationTest.entity
DetectedEntity robey.linguistics.sentenceanalysis.InterpretationTest.detectedEntity
List<DetectedEntity> robey.linguistics.sentenceanalysis.InterpretationTest.keywords
String robey.linguistics.sentenceanalysis.InterpretationTest.association
Map<Linguistics.SemanticRole, String> robey.linguistics.sentenceanalysis.InterpretationTest.semanticRoles
String robey.linguistics.sentenceanalysis.InterpretationTest.name
String robey.linguistics.sentenceanalysis.InterpretationTest.celebrity
boolean robey.linguistics.sentenceanalysis.InterpretationTest.isRoboy
String robey.linguistics.sentenceanalysis.InterpretationTest.objAnswer
String robey.linguistics.sentenceanalysis.InterpretationTest.predAnswer
RoboyEmotion robey.linguistics.sentenceanalysis.InterpretationTest.emotion
String robey.linguistics.sentenceanalysis.InterpretationTest.intent
```

```
String robpy.linguistics.sentenceanalysis.InterpretationTest.intentDistance
String robpy.linguistics.sentenceanalysis.InterpretationTest.parse
String robpy.linguistics.sentenceanalysis.InterpretationTest.parseAnswer
String robpy.linguistics.sentenceanalysis.InterpretationTest.underspecifiedTermQuestion
String robpy.linguistics.sentenceanalysis.InterpretationTest.underspecifiedQuestion
String robpy.linguistics.sentenceanalysis.InterpretationTest.underspecifiedAnswer
Linguistics.UtteranceSentiment robpy.linguistics.sentenceanalysis.InterpretationTest.s
String robpy.linguistics.sentenceanalysis.InterpretationTest.utteranceType
Linguistics.ParsingOutcome robpy.linguistics.sentenceanalysis.InterpretationTest.parsi
String robpy.linguistics.sentenceanalysis.InterpretationTest.answer
```

class

This state will:

- ask the interlocutor for his name
- query memory if the person is already known
- create and update the interlocutor in the context
- take one of two transitions: knownPerson or newPerson

IntroductionState interface: 1) Fallback is not required. 2) Outgoing transitions that have to be defined:

- knownPerson: following state if the person is already known
- newPerson: following state if the person is NOT known 3) No parameters are used.

Public Functions

```
robpy.dialog.states.ordinaryStates.IntroductionState.IntroductionState(String stateIdentifier)
Output robpy.dialog.states.ordinaryStates.IntroductionState.act()
Output robpy.dialog.states.ordinaryStates.IntroductionState.react(Interpretation input)
State robpy.dialog.states.ordinaryStates.IntroductionState.getNextState()
```

Protected Functions

```
Set<String> robpy.dialog.states.ordinaryStates.IntroductionState.getRequiredTransitions()
```

Private Functions

```
String robpy.dialog.states.ordinaryStates.IntroductionState.getNameFromInput(Interpretation input)
void robpy.dialog.states.ordinaryStates.IntroductionState.updateInterlocutorInContext(Interlocutor interlocutor)
String robpy.dialog.states.ordinaryStates.IntroductionState.getIntroPhrase()
String robpy.dialog.states.ordinaryStates.IntroductionState.getResponsePhrase(String name)
String robpy.dialog.states.ordinaryStates.IntroductionState.getRoboyFactsPhrase(Roboy robot)
```

Private Members

```
QJsonParser robey.dialog.states.ordinaryStates.IntroductionState.infoValues
final String robey.dialog.states.ordinaryStates.IntroductionState.UPDATE_KNOWN_PERSON =
final String robey.dialog.states.ordinaryStates.IntroductionState.LEARN_ABOUT_PERSON =
final Logger robey.dialog.states.ordinaryStates.IntroductionState.LOGGER = LogManager.getLogManager().getLogger(getClass().getName())
final String robey.dialog.states.ordinaryStates.IntroductionState.INFO_FILE_PARAMETER =
final RandomList<String> robey.dialog.states.ordinaryStates.IntroductionState.introPhrases
final RandomList<String> robey.dialog.states.ordinaryStates.IntroductionState.successResponses
final RandomList<String> robey.dialog.states.ordinaryStates.IntroductionState.failureResponses
Neo4jRelationship [] robey.dialog.states.ordinaryStates.IntroductionState.personPredictions
RandomList<Neo4jRelationship> robey.dialog.states.ordinaryStates.IntroductionState.robeyRelationships
RandomList<Neo4jProperty> robey.dialog.states.ordinaryStates.IntroductionState.robeyProperties
State robey.dialog.states.ordinaryStates.IntroductionState.nextState
```

```
class IO
    Helper class for IO related tasks.
```

Public Static Functions

```
static List<String> robey.util.IO.readLinesFromUtf8File(String path)
static MultiInputDevice robey.util.IO.getInputs(RosMainNode rosMainNode)
static MultiInputDevice robey.util.IO.getInputs(RosMainNode rosMainNode, String uuid)
static MultiOutputDevice robey.util.IO.getOutputs(RosMainNode rosMainNode)
static MultiOutputDevice robey.util.IO.getOutputs(RosMainNode rosMainNode, String uuid)
```

Private Static Attributes

```
final Logger robey.util.IO.logger = LogManager.getLogger()
```

```
class JsonEntryModel
```

Public Functions

```
RandomList<String> robey.util.JsonEntryModel.getQuestions()
Map<String, RandomList<String> > robey.util.JsonEntryModel.getAnswers()
Map<String, RandomList<String> > robey.util.JsonEntryModel.getFUP()
```

Package Attributes

```

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

```

Package Attributes

```

JsonEntryModel roboy.util.JsonModel.name
JsonEntryModel roboy.util.JsonModel.full_name
JsonEntryModel roboy.util.JsonModel.skills
JsonEntryModel roboy.util.JsonModel.abilities
JsonEntryModel roboy.util.JsonModel.age
JsonEntryModel roboy.util.JsonModel.future
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
JsonEntryModel roboy.util.JsonModel.IS
JsonEntryModel roboy.util.JsonModel.CHILD_OF
JsonEntryModel roboy.util.JsonModel.SIBLING_OF
JsonEntryModel roboy.util.JsonModel.OTHER
JsonEntryModel roboy.util.JsonModel.APPLES
JsonEntryModel roboy.util.JsonModel.ANIMAL
JsonEntryModel roboy.util.JsonModel.WORD
JsonEntryModel roboy.util.JsonModel.COLOR
JsonEntryModel roboy.util.JsonModel.PLANT
JsonEntryModel roboy.util.JsonModel.NAME
JsonEntryModel roboy.util.JsonModel.FRUIT
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<LexiconLiteral> roboy.memory.Lexicon.addTypeOfOwner(List< LexiconLiteral > resu  
List<LexiconLiteral> roboy.memory.Lexicon.scoreLiterals(List< LexiconLiteral > resul  
List<String> roboy.memory.Lexicon.getPermutations(String question)
```

Package Functions

```
String roboy.memory.Lexicon.bestLabelOf(String objlabel, String labell, String permuta
```

Private Functions

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

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 : public Comparable<LexiconLiteral>  
    An entity in the lexicon.
```

Public Functions

```
roboy.memory.LexiconLiteral.LexiconLiteral()  
roboy.memory.LexiconLiteral.LexiconLiteral(String URI, String label, String QuestionMa  
roboy.memory.LexiconLiteral.LexiconLiteral(String URI, String label, String QuestionMa  
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 : public Comparable<LexiconPredicate>
    A relation in the lexicon.
```

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

```
{ public int compare( LexiconPredicate lexpre1, LexiconPredicate lexpre2) { return lex-
pre1.compareTo(lexpre2); } } ]
```

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

Public Static Attributes

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

```
class Lists
    Helper class for list related tasks.
```

Public Static Functions

```
static List<String> roboy.util.Lists.stringList (String... strings)
```

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<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:

```
class MemoryIntegrationTest : public TestCase
```

Basically this class is a mirror copy of *Memory*'s Neo4JTest.

The exact same tests were used, only that instead of calling the functions in memory (few modifications), we call them via *Neo4jMemoryOperations*.

Public Functions

```
void roboy.memory.MemoryIntegrationTest.testCreateNode ()
```

```
void roboy.memory.MemoryIntegrationTest.testUpdateNode ()
```

```
void roboy.memory.MemoryIntegrationTest.testGetNode ()
```

```
void roboy.memory.MemoryIntegrationTest.testRemove ()
```

```
void roboy.memory.MemoryIntegrationTest.tearDown ()
```


Package Attributes

```
Gson roboy.memory.MemoryIntegrationTest.gson = new Gson()
long roboy.memory.MemoryIntegrationTest.timestamp = new Date().getTime()
final String roboy.memory.MemoryIntegrationTest.LUKAS = "{ 'label': 'Person', 'properties': { 'name': 'Lucas' } }"
final String roboy.memory.MemoryIntegrationTest.TOBY = "{ 'label': 'Person', 'properties': { 'name': 'Tobias' } }"
final String roboy.memory.MemoryIntegrationTest.ROBOY = "{ 'label': 'Robot', 'properties': { 'name': 'Roboy' } }"
```

class MemoryNodeModel

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

Subclassed by *roboy.memory.nodes.Interlocutor*, *roboy.memory.nodes.Roboy*

Public Functions

```
roboy.memory.nodes.MemoryNodeModel.MemoryNodeModel (Neo4jMemoryInterface memory)
roboy.memory.nodes.MemoryNodeModel.MemoryNodeModel (String jsonString, Neo4jMemoryInterface memory)
roboy.memory.nodes.MemoryNodeModel.MemoryNodeModel (boolean stripQuery, Neo4jMemoryInterface memory)
int roboy.memory.nodes.MemoryNodeModel.getId ()
void roboy.memory.nodes.MemoryNodeModel.setId (int id)
ArrayList<Neo4jLabel> roboy.memory.nodes.MemoryNodeModel.getLabels ()
void roboy.memory.nodes.MemoryNodeModel.setLabel (Neo4jLabel label)
HashMap<Neo4jProperty, Object> roboy.memory.nodes.MemoryNodeModel.getProperties ()
Object roboy.memory.nodes.MemoryNodeModel.getProperty (Neo4jProperty key)
void roboy.memory.nodes.MemoryNodeModel.setProperties (HashMap< Neo4jProperty , Object> properties)
void roboy.memory.nodes.MemoryNodeModel.setProperty (Neo4jProperty key, Object property)
HashMap<Neo4jRelationship, ArrayList<Integer> > roboy.memory.nodes.MemoryNodeModel.getRelationships ()
ArrayList<Integer> roboy.memory.nodes.MemoryNodeModel.getRelationship (Neo4jRelationship key)
void roboy.memory.nodes.MemoryNodeModel.setRelationships (HashMap< Neo4jRelationship , ArrayList<Integer> > relationships)
void roboy.memory.nodes.MemoryNodeModel.setRelationships (Neo4jRelationship key, ArrayList<Integer> values)
void roboy.memory.nodes.MemoryNodeModel.setRelationship (Neo4jRelationship key, Integer value)
void roboy.memory.nodes.MemoryNodeModel.setStripQuery (boolean strip)
String roboy.memory.nodes.MemoryNodeModel.toJSON ()
    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.
String roboy.memory.nodes.MemoryNodeModel.toString ()
```

Protected Attributes

```
Neo4jMemoryInterface roboy.memory.nodes.MemoryNodeModel.memory
```

Package Attributes

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

Package Static Attributes

```
final Logger roboy.memory.nodes.MemoryNodeModel.logger = LogManager.getLogger()
```

Private Members

```
int roboy.memory.nodes.MemoryNodeModel.id
ArrayList<String> roboy.memory.nodes.MemoryNodeModel.labels
Neo4jLabel roboy.memory.nodes.MemoryNodeModel.label
HashMap<String, Object> roboy.memory.nodes.MemoryNodeModel.properties
HashMap<String, ArrayList<Integer> > roboy.memory.nodes.MemoryNodeModel.relationships
```

class

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.

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 MiniTestStateMachineCreator

Helper class for testing: creates a minimal state machine with 2 states.

Public Static Functions

```
static String roboy.dialog.MiniTestStateMachineCreator.getMiniStateMachineString()
Returns a String representation of a minimal state machine with only two states.
```

The representation is equal to the machine from *getMiniStateMachine()*.

Return String representation of a minimal state machine with only two states

```
static DialogStateMachine roboy.dialog.MiniTestStateMachineCreator.getMiniStateMachine()
Creates a minimal state machine with only two states from code.
```

The machine is equal to the String representation from *getMiniStateMachineString()*.

Return a minimal state machine with only two states created from code

class

Meta class to combine multiple input devices.

Public Functions

```
roboy.io.MultiInputDevice.MultiInputDevice(InputDevice mainInput)
void roboy.io.MultiInputDevice.addInputDevice(InputDevice additionalInput)
Input roboy.io.MultiInputDevice.listen()
void roboy.io.MultiInputDevice.cleanup()
    Calls cleanup() for every included input device that implements CleanUp.
void roboy.io.MultiInputDevice.finalize()
```

Private Members

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

class

Meta class to combine multiple output devices.

Public Functions

```
roboy.io.MultiOutputDevice.MultiOutputDevice(OutputDevice device)
void roboy.io.MultiOutputDevice.add(OutputDevice additionalDevice)
void roboy.io.MultiOutputDevice.act(List< Action > actions)
void roboy.io.MultiOutputDevice.cleanup()
    Calls cleanup for all devices and removes them from the devices list after cleaning.
void roboy.io.MultiOutputDevice.finalize()
```

Private Members

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

enum Neo4jLabel

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.Neo4jLabel.Neo4jLabel(String type)
```

Public Members

```
roboy.memory.Neo4jLabel.Person = ("Person")
roboy.memory.Neo4jLabel.TelegramPerson = ("TelegramPerson")
roboy.memory.Neo4jLabel.FacebookPerson = ("FacebookPerson")
roboy.memory.Neo4jLabel.SlackPerson = ("SlackPerson")
roboy.memory.Neo4jLabel.Robot = ("Robot")
roboy.memory.Neo4jLabel.Company = ("Company")
roboy.memory.Neo4jLabel.University = ("University")
roboy.memory.Neo4jLabel.City = ("City")
roboy.memory.Neo4jLabel.Country = ("Country")
roboy.memory.Neo4jLabel.Hobby = ("Hobby")
roboy.memory.Neo4jLabel.Occupation = ("Occupation")
roboy.memory.Neo4jLabel.Object = ("Object")
roboy.memory.Neo4jLabel.Location = ("Location")
roboy.memory.Neo4jLabel.Organization = ("Organization")
roboy.memory.Neo4jLabel.Other = ("Other")
roboy.memory.Neo4jLabel.None = ("")
String roboy.memory.Neo4jLabel.type
```

Public Static Functions

```
roboy.memory.Neo4jLabel.[static initializer]()
static Neo4jLabel roboy.memory.Neo4jLabel.lookupByType(String type)
static boolean roboy.memory.Neo4jLabel.contains(String type)
```

Private Static Attributes

```
Maps.newHashMapWithExpectedSize(Neo4jLabel.values().length) ]
```

class

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

Public Functions

```
roboy.memory.Neo4jMemory.Neo4jMemory()
boolean roboy.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.

String `roboy.memory.Neo4jMemory.getId(int id)`

This query retrieves a single node by its ID.

Return Node representation of the result.

Parameters

- `id`: the ID of requested

ArrayList<Integer> `roboy.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 `roboy.memory.Neo4jMemory.create(MemoryNodeModel query)`

boolean `roboy.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.

Private Members

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

Private Static Attributes

final Logger `roboy.memory.Neo4jMemory.logger` = LogManager.getLogger()

interface `Neo4jMemoryInterface`

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

Subclassed by *roboy.memory.DummyMemory*, *roboy.memory.Neo4jMemory*

Public Functions

boolean `roboy.memory.Neo4jMemoryInterface.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.

String `roboy.memory.Neo4jMemoryInterface.getId(int id)`

This query retrieves a single node by its ID.

Return String with node representation of the result.

Parameters

- `id`: the ID of requested

ArrayList<Integer> roboy.memory.Neo4jMemoryInterface.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 roboy.memory.Neo4jMemoryInterface.create (MemoryNodeModel query)

boolean roboy.memory.Neo4jMemoryInterface.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.

class Neo4jMemoryOperations

This Class creates an interface to connect to memory.

Instead of calling via a service via ROS, we simply call the function directly and get returned a JSON string.

Public Static Functions

static String roboy.memory.Neo4jMemoryOperations.get (String query)

Get the Node ID.

Return JSON containing ID of node

Parameters

- `query`: Query to specify Node to get. Ex: `{ "labels": ["Person"], "label": "Person", "properties": { "name": "davis" } }`

static String roboy.memory.Neo4jMemoryOperations.cypher (String query)

Cypher Method that is never called TODO: Implement this feature or refactor it out, it's kind of here because there was a service.

Return

Parameters

- `query`:

static String roboy.memory.Neo4jMemoryOperations.create (String query)

Create a node.

Return JSON containing the ID of the new node

Parameters

- `query`: Query with data regarding the node. Ex: `{ "labels": ["Organization"], "label": "Organization", "properties": { "name": "korn" } }`

static String roboy.memory.Neo4jMemoryOperations.update(String query)
Update Nodes.

Return JSON establishing whether or not the connection was made or not

Parameters

- **query:** Query to link two nodes together. Ex: {"labels":["Person"],"label":"Person","properties":{"name":"davis"},"relationships":{"FROM":[369]},"id":368}

static String roboy.memory.Neo4jMemoryOperations.delete(String query)
Delete a Node.

Return Whether or not deleting was successful or not

Parameters

- **query:** JSON query to delete a specified node. Ex: {'type':'node','id':361,'properties_list':['sex'],'relationships':{'FRIEND_OF':[426]}}

enum Neo4jProperty

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.Neo4jProperty.Neo4jProperty(String type)

Public Members

```
roboy.memory.Neo4jProperty.name = ("name")
roboy.memory.Neo4jProperty.sex = ("sex")
roboy.memory.Neo4jProperty.full_name = ("full_name")
roboy.memory.Neo4jProperty.age = ("age")
roboy.memory.Neo4jProperty.skills = ("skills")
roboy.memory.Neo4jProperty.abilities = ("abilities")
roboy.memory.Neo4jProperty.future = ("future")
roboy.memory.Neo4jProperty.birthdate = ("birthdate")
roboy.memory.Neo4jProperty.facebook_id = ("facebook_id")
roboy.memory.Neo4jProperty.telegram_id = ("telegram_id")
roboy.memory.Neo4jProperty.slack_id = ("slack_id")
roboy.memory.Neo4jProperty.whatsapp_id = ("whatsapp_id")
roboy.memory.Neo4jProperty.line_id = ("line_id")
String roboy.memory.Neo4jProperty.type
```

Public Static Functions

```
roboy.memory.Neo4jProperty.[static initializer]()  
static Neo4jProperty roboy.memory.Neo4jProperty.lookupByType(String type)  
static boolean roboy.memory.Neo4jProperty.contains(String type)
```

Private Static Attributes

```
Maps.newHashMapWithExpectedSize(Neo4jProperty.values().length) ]
```

enum Neo4jRelationship

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.Neo4jRelationship.Neo4jRelationship(String type)
```

Public Members

```
roboy.memory.Neo4jRelationship.FROM = ("FROM")  
roboy.memory.Neo4jRelationship.HAS_HOBBY = ("HAS_HOBBY")  
roboy.memory.Neo4jRelationship.LIVE_IN = ("LIVE_IN")  
roboy.memory.Neo4jRelationship.STUDY_AT = ("STUDY_AT")  
roboy.memory.Neo4jRelationship.OCCUPIED_AS = ("OCCUPIED_AS")  
roboy.memory.Neo4jRelationship.WORK_FOR = ("WORK_FOR")  
roboy.memory.Neo4jRelationship.FRIEND_OF = ("FRIEND_OF")  
roboy.memory.Neo4jRelationship.MEMBER_OF = ("MEMBER_OF")  
roboy.memory.Neo4jRelationship.CHILD_OF = ("CHILD_OF")  
roboy.memory.Neo4jRelationship.SIBLING_OF = ("SIBLING_OF")  
roboy.memory.Neo4jRelationship.KNOW = ("KNOW")  
roboy.memory.Neo4jRelationship.OTHER = ("OTHER")  
roboy.memory.Neo4jRelationship.IS = ("IS")  
String roboy.memory.Neo4jRelationship.type
```

Public Static Functions

```
static Neo4jLabel roboy.memory.Neo4jRelationship.determineNodeType(Neo4jRelationship r  
roboy.memory.Neo4jRelationship.[static initializer]()  
static Neo4jRelationship roboy.memory.Neo4jRelationship.lookupByType(String type)  
static boolean roboy.memory.Neo4jRelationship.contains(String type)
```


Private Static Attributes

```
Maps.newHashMapWithExpectedSize(Neo4jRelationship.values().length) ]
```

```
template <V>
```

```
class ObservableValue : public Observable, public roboy::context::AbstractValue<V>
```

A *Value* that supports adding Observers.

These will be notified whenever a new value is added.

Parameters

- <V>:

Subclassed by *roboy.context.contextObjects.FaceCoordinates*

Public Functions

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

```
synchronized void roboy.context.ObservableValue< V >.updateValue(V value)
```

Private Members

```
V roboy.context.ObservableValue< V >.value = null
```

```
class
```

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

Public Functions

```
roboy.linguistics.sentenceanalysis.OntologyNERAnalyzer.OntologyNERAnalyzer()
```

```
Interpretation roboy.linguistics.sentenceanalysis.OntologyNERAnalyzer.analyze(Interpre-
```

Private Members

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

```
class
```

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.

Public Functions

```
roboy.linguistics.sentenceanalysis.OpenNLPParser.OpenNLPParser()
```

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

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

Public Static Functions

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

Private Functions

```
Interpretation roboy.linguistics.sentenceanalysis.OpenNLPParser.extractPAS(Interpretation)
Map<SemanticRole, String> roboy.linguistics.sentenceanalysis.OpenNLPParser.top(Parse p)
Map<SemanticRole, String> roboy.linguistics.sentenceanalysis.OpenNLPParser.sbar(Parse p)
Map<SemanticRole, String> roboy.linguistics.sentenceanalysis.OpenNLPParser.vp(Parse p)
```

Private Members

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

Public Functions

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

Private Static Attributes

```
final OpenNLPParser roboy.linguistics.sentenceanalysis.OpenNLPParserTest.parser = new OpenNLPParser()
class
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.
```

Public Functions

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

Private Functions

```
String [] roboy.linguistics.sentenceanalysis.OpenNLPPPOSTagger.extractPosTag(List< String>)
```

Private Members

POSTaggerME `robey.linguistics.sentenceanalysis.OpenNLPPPOSTagger.tagger`

class OtherQuestionsUpdater : public *robey::context::InternalUpdater*<*AbstractValue*<Integer>, Integer>
Updater available to all DM for adding new values to the *DialogTopics* attribute.

Public Functions

`robey.context.contextObjects.OtherQuestionsUpdater.OtherQuestionsUpdater (AbstractValue`

class Output

Output static inner class represents the return values of *act()* and *react()* methods.

There are four possible scenarios:

- the state wants to say something -> a single interpretation is returned
- the state does not say anything -> no interpretation
- the state does not know how to react -> fallback state is required to fix this
- the state wants to end the conversation -> reset the whole dialog state machine

To create an instance of this class inside the *act()* or *react()* method use following:

- *Output.say*(new Interpretation(...)) - to return an interpretation
- *Output.say*("Some phrase here") - to return an interpretation (will be created from string)
- *Output.sayNothing()* - to make clear that you don't want to say anything
- *Output.useFallback()* - to indicate that you can't react and want to use the fallback
- *Output.endConversation()* - to stop the conversation immediately and reset the state machine
- *Output.endConversation*("last words") - to say the last words and reset the state machine afterwards

Public Functions

boolean `robey.dialog.states.definitions.State.Output.hasInterpretation()`

boolean `robey.dialog.states.definitions.State.Output.requiresFallback()`

boolean `robey.dialog.states.definitions.State.Output.isEmpty()`

boolean `robey.dialog.states.definitions.State.Output.isEndOfConversation()`

Interpretation `robey.dialog.states.definitions.State.Output.getInterpretation()`

Output `robey.dialog.states.definitions.State.Output.setSegue (Segue s)`

A segue is a smooth transition from one topic to the next.

You can add a segues of a specific types to the state output if you want to change the topic. Segues have a certain probabilities to be used and are always added after the original output was said.

Return the same *Output* object so you can chain multiple function calls on it

Parameters

- s: segue to add

boolean `robey.dialog.states.definitions.State.Output.hasSegue()`

```
Segue roboy.dialog.states.definitions.State.Output.getSegue()  
Output roboy.dialog.states.definitions.State.Output.setEmotion(String emotion)  
boolean roboy.dialog.states.definitions.State.Output.hasEmotion()  
String roboy.dialog.states.definitions.State.Output.getEmotion()
```

Public Static Functions

static Output roboy.dialog.states.definitions.State.Output.say(Interpretation i)
Say a phrase.

Return *State.Output* object with appropriate settings

Parameters

- i: interpretation object that contains the phrase

static Output roboy.dialog.states.definitions.State.Output.say(String s)
Say a phrase.

Return *State.Output* object with appropriate settings

Parameters

- s: phrase to say

static Output roboy.dialog.states.definitions.State.Output.sayNothing()
Say nothing (as the output of *State* act/react).

Return *State.Output* object with appropriate settings

static Output roboy.dialog.states.definitions.State.Output.useFallback()
Indicate that current state has no idea how to react and that the dialog system should use a fallback state to react instead.

This option is only allowed for react(...) output. States should never use this option from the *act()* function.

Return *State.Output* object with appropriate settings

static Output roboy.dialog.states.definitions.State.Output.endConversation()
End the conversation immediately.

Return *State.Output* object with appropriate settings

static Output roboy.dialog.states.definitions.State.Output.endConversation(String lastWords)
End the conversation after saying some last words.

Return *State.Output* object with appropriate settings

Parameters

- lastWords: last word to say (something like “I’ll be back”)

Private Functions

roboy.dialog.states.definitions.State.Output.Output (OutputType type, Interpretation interpretation)
Private constructor, used only inside static methods.

Parameters

- type: type of this react object
- interpretation: optional interpretation object (or null)

Private Members

```
final Logger roboy.dialog.states.definitions.State.Output.logger = LogManager.getLogger()
final OutputType roboy.dialog.states.definitions.State.Output.type
final Interpretation roboy.dialog.states.definitions.State.Output.interpretation
Segue roboy.dialog.states.definitions.State.Output.segue
String roboy.dialog.states.definitions.State.Output.emotion
```

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.FreeTTSPutput*, *roboy.io.IBMWatsonOutput*, *roboy.io.MultiOutputDevice*, *roboy.io.TelegramOutput*, *roboy.io.UdpOutput*

Public Functions

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

```
enum OutputType
```

Public Members

```
roboy.dialog.states.definitions.State.Output.OutputType.INTERPRETATION
roboy.dialog.states.definitions.State.Output.OutputType.SAY_NOTHING
roboy.dialog.states.definitions.State.Output.OutputType.USE_FALLBACK
roboy.dialog.states.definitions.State.Output.OutputType.END_CONVERSATION
```

```
template <KEY, VALUE>
```

```
class Pair
```

A simple tuple class to store two values.

Like javafx.util.pair just more simple, but accessible from maven.

Parameters

- <KEY>: First value of the tuple
- <VALUE>: Second value of the tuple

Public Functions

```
roboy.util.Pair< KEY, VALUE >.Pair(KEY key, VALUE value)
KEY roboy.util.Pair< KEY, VALUE >.getKey()
VALUE roboy.util.Pair< KEY, VALUE >.getValue()
```

Private Members

```
KEY roboy.util.Pair< KEY, VALUE >.key = null
VALUE roboy.util.Pair< KEY, VALUE >.value = null
enum ParsingOutcome
```

Public Members

```
roboy.linguistics.Linguistics.ParsingOutcome.SUCCESS
roboy.linguistics.Linguistics.ParsingOutcome.FAILURE
roboy.linguistics.Linguistics.ParsingOutcome.UNDERSPECIFIED
class
    Passive state to start a conversation.
    Roboy is waiting until a greeting or his name is detected.
```

Public Functions

```
roboy.dialog.states.ordinaryStates.PassiveGreetingsState.PassiveGreetingsState(String
Output roboy.dialog.states.ordinaryStates.PassiveGreetingsState.act()
Output roboy.dialog.states.ordinaryStates.PassiveGreetingsState.react(Interpretation i
State roboy.dialog.states.ordinaryStates.PassiveGreetingsState.getNextState()
boolean roboy.dialog.states.ordinaryStates.PassiveGreetingsState.isFallbackRequired()
```

Protected Functions

```
Set<String> roboy.dialog.states.ordinaryStates.PassiveGreetingsState.getRequiredTransi
Set<String> roboy.dialog.states.ordinaryStates.PassiveGreetingsState.getRequiredParamet
```

Private Members

```
final String roboy.dialog.states.ordinaryStates.PassiveGreetingsState.TRANSITION_GREET
State roboy.dialog.states.ordinaryStates.PassiveGreetingsState.next
template <Target>
```

class

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

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

Parameters

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

Public Functions

roboy.context.PeriodicUpdater< Target >.PeriodicUpdater(Target target)

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

Parameters

- target: The target attribute of the update() method.

Public Static Attributes

int roboy.context.PeriodicUpdater< Target >.updateFrequencySeconds = 1

Protected Attributes

final Target roboy.context.PeriodicUpdater< Target >.target

Private Functions

void roboy.context.PeriodicUpdater< Target >.start()

Starts the ScheduledExecutorService of the updating thread.

Private Members

final ScheduledExecutorService roboy.context.PeriodicUpdater< Target >.scheduler = Executors.newSingleThreadExecutor()

class

Personal Information Asking State.

The state tries to interact with the Interlocutor to learn new information about the person. This information is sent to the Roboy Memory Module through Neo4jMemoryInterface for storing. Afterwards, Roboy can use this acquired data for the future interactions with the same person.

- if there is no existing Interlocutor or the data is missing, ask a question
- the question topic (intent) is selected from the Neo4jRelationship predicates
- retrieve the questions stored in the QAList json file
- update the Context IntentsHistory
- try to extract the result from the Interpretation
- retrieve the answers stored in the QAList json file

- send the result to Memory

PersonalInformationAskingState interface: 1) Fallback is not required. 2) Outgoing transitions that have to be defined:

- TRANSITION_INFO_OBTAINED: following state if the question was asked 3) Required parameters: path to the QAList.json file.

Public Functions

```
roboy.dialog.states.ordinaryStates.PersonalInformationAskingState.PersonalInformationA
Output roboy.dialog.states.ordinaryStates.PersonalInformationAskingState.act ()
Output roboy.dialog.states.ordinaryStates.PersonalInformationAskingState.react (Interpr
State roboy.dialog.states.ordinaryStates.PersonalInformationAskingState.getNextState ()
```

Public Static Attributes

```
final String roboy.dialog.states.ordinaryStates.PersonalInformationAskingState.INTENTS.
```

Protected Functions

```
Set<String> roboy.dialog.states.ordinaryStates.PersonalInformationAskingState.getRequi
Set<String> roboy.dialog.states.ordinaryStates.PersonalInformationAskingState.getRequi.
```

Package Attributes

```
final Logger roboy.dialog.states.ordinaryStates.PersonalInformationAskingState.LOGGER =
```

Private Functions

```
String roboy.dialog.states.ordinaryStates.PersonalInformationAskingState.InferResult (I
```

Private Members

```
QAJsonParser roboy.dialog.states.ordinaryStates.PersonalInformationAskingState.qaValue
Neo4jRelationship [] roboy.dialog.states.ordinaryStates.PersonalInformationAskingState
Neo4jRelationship roboy.dialog.states.ordinaryStates.PersonalInformationAskingState.se
State roboy.dialog.states.ordinaryStates.PersonalInformationAskingState.nextState
final String roboy.dialog.states.ordinaryStates.PersonalInformationAskingState.TRANSIT
final String roboy.dialog.states.ordinaryStates.PersonalInformationAskingState.QA_FILE
```


class

Personal Information Asking State.

The state tries to interact with the Interlocutor to learn new information about the person. This information is sent to the Roboy Memory Module through Neo4jMemoryInterface for storing. Afterwards, Roboy can use this acquired data for the future interactions with the same person.

- if there is no existing Interlocutor or the data is missing, ask a question
- the question topic (intent) is selected from the Neo4jRelationship predicates
- retrieve the questions stored in the QAList json file
- update the Context IntentsHistory
- try to extract the result from the Interpretation
- retrieve the answers stored in the QAList json file
- send the result to Memory

PersonalInformationAskingState interface: 1) Fallback is not required. 2) Outgoing transitions that have to be defined, following state if the question was asked:

- skills,
- abilities,
- robpy. 3) Required parameters: path to the QAList.json file.

Public Functions

```
robpy.dialog.states.expoStates.PersonalInformationAskingState.PersonalInformationAskingState()
Output robpy.dialog.states.expoStates.PersonalInformationAskingState.act()
Output robpy.dialog.states.expoStates.PersonalInformationAskingState.react(Interpretation)
State robpy.dialog.states.expoStates.PersonalInformationAskingState.getNextState()
```

Public Static Attributes

```
final String robpy.dialog.states.expoStates.PersonalInformationAskingState.INTENTS_HISTORY
```

Protected Functions

```
Set<String> robpy.dialog.states.expoStates.PersonalInformationAskingState.getRequiredTopics()
Set<String> robpy.dialog.states.expoStates.PersonalInformationAskingState.getRequiredPredicates()
```

Private Functions

```
String robpy.dialog.states.expoStates.PersonalInformationAskingState.InferResult(Interpretation)
```

Private Members

```
final String [] roboy.dialog.states.expoStates.PersonalInformationAskingState.TRANSITION
final String [] roboy.dialog.states.expoStates.PersonalInformationAskingState.INTENT_N
final String roboy.dialog.states.expoStates.PersonalInformationAskingState.QA_FILE_PAR
final Logger roboy.dialog.states.expoStates.PersonalInformationAskingState.LOGGER = LogM
QAJsonParser roboy.dialog.states.expoStates.PersonalInformationAskingState.qaValues
Neo4jRelationship [] roboy.dialog.states.expoStates.PersonalInformationAskingState.pre
Neo4jRelationship roboy.dialog.states.expoStates.PersonalInformationAskingState.select
int roboy.dialog.states.expoStates.PersonalInformationAskingState.otherIndex
State roboy.dialog.states.expoStates.PersonalInformationAskingState.nextState
```

class

Personal Information Update State.

This state is only entered if there are some known facts about the active interlocutor. The state tries to interact with the Interlocutor to update the existing information about the person. This information is sent to the Roboy Memory Module through Neo4jMemoryInterface to keep it up to date.

- if there is an existing entry under a specific Neo4jRelationship predicate, select the predicate
- check the Context IntentsHistory if we already asked similar questions
- the question topic (intent) is selected upon the predicate
- update the Context IntentsHistory with the selected predicate
- retrieve the follow-up questions stored in the QAList json file
- retrieve the follow-up answers stored in the QAList json file

PersonalInformationFollowUpState interface: 1) Fallback is not required. 2) Outgoing transitions that have to be defined:

- TRANSITION_INFO_UPDATED: following state if the question was asked 3) Required parameters: path to the QAList.json file.

Public Functions

```
roboy.dialog.states.ordinaryStates.PersonalInformationFollowUpState.PersonalInformation
Output roboy.dialog.states.ordinaryStates.PersonalInformationFollowUpState.act ()
Output roboy.dialog.states.ordinaryStates.PersonalInformationFollowUpState.react (Interp
State roboy.dialog.states.ordinaryStates.PersonalInformationFollowUpState.getNextState
```

Public Static Attributes

```
final String roboy.dialog.states.ordinaryStates.PersonalInformationFollowUpState.INTENT
```

Protected Functions

`Set<String> robby.dialog.states.ordinaryStates.PersonalInformationFollowUpState.getReq`

`Set<String> robby.dialog.states.ordinaryStates.PersonalInformationFollowUpState.getReq`

Private Functions

`String robby.dialog.states.ordinaryStates.PersonalInformationFollowUpState.InferUpdate`

Private Members

`QAJsonParser robby.dialog.states.ordinaryStates.PersonalInformationFollowUpState.qaVal`

`Neo4jRelationship [] robby.dialog.states.ordinaryStates.PersonalInformationFollowUpSta`

`Neo4jRelationship robby.dialog.states.ordinaryStates.PersonalInformationFollowUpState.`

`State robby.dialog.states.ordinaryStates.PersonalInformationFollowUpState.nextState`

`final String robby.dialog.states.ordinaryStates.PersonalInformationFollowUpState.TRANS`

`final String robby.dialog.states.ordinaryStates.PersonalInformationFollowUpState.QA_FI`

`final Logger robby.dialog.states.ordinaryStates.PersonalInformationFollowUpState.LOGGE`

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.

Subclassed by *robby.dialog.personality.StateBasedPersonality*

Public Functions

`List<Action> robby.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

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 *robby.linguistics.phonetics.DoubleMetaphoneEncoder*, *robby.linguistics.phonetics.MetaphoneEncoder*, *robby.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 PhraseCollection
```

A (temporary) central class to store short lists of phrases.

The lists are stored in separate files. This class loads all of them once at the beginning, so all the lists can be used by any other class later.

We might define a JSON format to replace the single files later.

Public Static Attributes

```
RandomList<String> roboy.talk.PhraseCollection.CONNECTING_PHRASES = readFile("resources/phraseLi
RandomList<String> roboy.talk.PhraseCollection.QUESTION_ANSWERING_REENTERING = readFile("res
RandomList<String> roboy.talk.PhraseCollection.QUESTION_ANSWERING_START = readFile("resources/
RandomList<String> roboy.talk.PhraseCollection.SEGUE_AVOID_ANSWER = readFile("resources/phraseLi
RandomList<String> roboy.talk.PhraseCollection.SEGUE_DISTRACT = readFile("resources/phraseLists/seg
RandomList<String> roboy.talk.PhraseCollection.SEGUE_FLATTERY = readFile("resources/phraseLists/seg
RandomList<String> roboy.talk.PhraseCollection.SEGUE_JOBS = readFile("resources/phraseLists/segue-job
RandomList<String> roboy.talk.PhraseCollection.SEGUE_PICKUP = FileLineReader.readFile("resources/ph
RandomList<String> roboy.talk.PhraseCollection.OFFER_GAME_PHRASES = readFile("resources/phraseLi
RandomList<String> roboy.talk.PhraseCollection.AKINATOR_INTRO_PHRASES = readFile("resources/phr
RandomList<String> roboy.talk.PhraseCollection.ROBOY_WINNER_PHRASES = readFile("resources/phras
RandomList<String> roboy.talk.PhraseCollection.ROBOY_LOSER_PHRASES = readFile("resources/phraseL
RandomList<String> roboy.talk.PhraseCollection.GAME Asking_PHRASES = readFile("resources/phraseL
RandomList<String> roboy.talk.PhraseCollection.OFFER_FILTER_PHRASES = readFile("resources/phras
RandomList<String> roboy.talk.PhraseCollection.AKINATOR_ERROR_PHRASES = readFile("resources/phr
```

```

RandomList<String> roboy.talk.PhraseCollection.PROFANITY_CHECK_WORDS = FileLineReader.readFile(
RandomList<String> roboy.talk.PhraseCollection.FACTS = readFile("resources/phraseLists/expoPhrases/fac
RandomList<String> roboy.talk.PhraseCollection.INFO_ROBOY_INTENT_PHRASES = readFile("resource
RandomList<String> roboy.talk.PhraseCollection.JOKES = readFile("resources/phraseLists/expoPhrases/jok
RandomList<String> roboy.talk.PhraseCollection.NEGATIVE_SENTIMENT_PHRASES = readFile("resourc
RandomList<String> roboy.talk.PhraseCollection.OFFER_FACTS_PHRASES = readFile("resources/phraseL
RandomList<String> roboy.talk.PhraseCollection.OFFER_FAMOUS_ENTITIES_PHRASES = readFile("res
RandomList<String> roboy.talk.PhraseCollection.OFFER_JOKES_PHRASES = readFile("resources/phraseL
RandomList<String> roboy.talk.PhraseCollection.OFFER_MATH_PHRASES = readFile("resources/phraseLi
RandomList<String> roboy.talk.PhraseCollection.PARSER_ERROR = readFile("resources/phraseLists/expoF

```

class

Corrects abbreviated forms like "I'm" to complete forms like "I am" which are expected by later sentence analyses.

Public Functions

```

Interpretation roboy.linguistics.sentenceanalysis.Preprocessor.analyze(Interpretation

```

class

Checks for words and stores if the sentence has profanity in the *Interpretation* Profanity feature that is later read out and fed to the output module.

Public Functions

```

Interpretation roboy.linguistics.sentenceanalysis.ProfanityAnalyzer.analyze(Interpretation

```

class QAFileParser

Parses files containing predefined questions and answers.

Expects the following input pattern: { "INTENT": { "Q": ["Question phrasing 1", "Question phrasing 2", "Question phrasing 3"], "A": { "SUCCESS": ["Possible answer on success 1", "Possible answer on success 2"], "FAILURE": ["Possible answer on failure"] } } }

See more examples in resources/sentences

Public Functions

```

roboy.util.QAFileParser.QAFileParser(String file)
Map<String, List<String> > roboy.util.QAFileParser.getQuestions()
Map<String, List<String> > roboy.util.QAFileParser.getSuccessAnswers()
Map<String, List<String> > roboy.util.QAFileParser.getFailureAnswers()

```

Private Members

```

final Logger roboy.util.QAFileParser.logger = LogManager.getLogger()

```

Private Static Attributes

```
Map<String, List<String> > roboy.util.QAFileParser.questions
Map<String, List<String> > roboy.util.QAFileParser.successAnswers
Map<String, List<String> > roboy.util.QAFileParser.failureAnswers
```

class QAJsonParser

Getting values for personalStates and follow-up questions from a JSON file.

Public Functions

```
roboy.util.QAJsonParser.QAJsonParser()
roboy.util.QAJsonParser.QAJsonParser(String file)
JsonModel roboy.util.QAJsonParser.parse(String file)
JsonModel roboy.util.QAJsonParser.getQA()
JsonEntryModel roboy.util.QAJsonParser.getEntry(Neo4jRelationship relationship)
JsonEntryModel roboy.util.QAJsonParser.getEntry(Neo4jProperty property)
RandomList<String> roboy.util.QAJsonParser.getQuestions(Neo4jRelationship relationship)
Map<String, RandomList<String> > roboy.util.QAJsonParser.getAnswers(Neo4jRelationship relationship)
RandomList<String> roboy.util.QAJsonParser.getSuccessAnswers(Neo4jRelationship relationship)
RandomList<String> roboy.util.QAJsonParser.getFailureAnswers(Neo4jRelationship relationship)
Map<String, RandomList<String> > roboy.util.QAJsonParser.getFollowUp(Neo4jRelationship relationship)
RandomList<String> roboy.util.QAJsonParser.getFollowUpQuestions(Neo4jRelationship relationship)
RandomList<String> roboy.util.QAJsonParser.getFollowUpAnswers(Neo4jRelationship relationship)
RandomList<String> roboy.util.QAJsonParser.getQuestions(Neo4jProperty property)
Map<String, RandomList<String> > roboy.util.QAJsonParser.getAnswers(Neo4jProperty property)
RandomList<String> roboy.util.QAJsonParser.getSuccessAnswers(Neo4jProperty property)
RandomList<String> roboy.util.QAJsonParser.getFailureAnswers(Neo4jProperty property)
Map<String, RandomList<String> > roboy.util.QAJsonParser.getFollowUp(Neo4jProperty property)
RandomList<String> roboy.util.QAJsonParser.getFollowUpQuestions(Neo4jProperty property)
RandomList<String> roboy.util.QAJsonParser.getFollowUpAnswers(Neo4jProperty property)
```

Private Functions

```
JsonEntryModel roboy.util.QAJsonParser.getJsonEntryModel(String type)
```

Private Members

```
Gson roboy.util.QAJsonParser.gson
JsonModel roboy.util.QAJsonParser.jsonObject
final Logger roboy.util.QAJsonParser.LOGGER = LogManager.getLogger()
class QAParserTest
```

Public Functions

```
void roboy.parser.QAParserTest.testEntryEquivalency ()
void roboy.parser.QAParserTest.testQuestions ()
void roboy.parser.QAParserTest.testAnswers ()
void roboy.parser.QAParserTest.testFollowUp ()
```

Public Static Functions

```
static void roboy.parser.QAParserTest.createJsonAndParse ()
static void roboy.parser.QAParserTest.cleanUpJson ()
```

Package Static Attributes

```
QAJsonParser roboy.parser.QAParserTest.parser
String roboy.parser.QAParserTest.path = "test.json"
class
```

This state will answer generalStates questions.

The parser:

- provides triples generated from the question
- adds the answer to the question if there is an answer in DBpedia
- tells a specifying followup question if the interlocutor's question was ambiguous

This state:

- checks if interlocutor wants to play a game
- returns the answer if provided by the parser
- asks the specifying followup question if provided by the parser
- - if answered with yes > will use the parser again to get the answer to the original question
 - if answered with no > will use a segue to avoid answer
- tries to query memory if there is no answer to the question
- queries the fallback if memory fails to answer as well

QuestionAnsweringState interface: 1) Fallback is required. 2) Outgoing transitions that have to be defined:

- finishedQuestionAnswering: following state if this state if finished with answering questions 3) No parameters are used.

Public Functions

```
robey.dialog.states.ordinaryStates.QuestionAnsweringState.QuestionAnsweringState (String)
Output robey.dialog.states.ordinaryStates.QuestionAnsweringState.act ()
Output robey.dialog.states.ordinaryStates.QuestionAnsweringState.react (Interpretation
State robey.dialog.states.ordinaryStates.QuestionAnsweringState.getNextState ()
boolean robey.dialog.states.ordinaryStates.QuestionAnsweringState.isFallbackRequired ()
```

Protected Functions

```
Set<String> robey.dialog.states.ordinaryStates.QuestionAnsweringState.getRequiredTrans
```

Private Functions

```
Output robey.dialog.states.ordinaryStates.QuestionAnsweringState.reactToSpecifyingAnswer
React to answer of the specifying question asked previously.
```

Return answer to the answer to the original question if specifying question was answered with ‘yes’

Parameters

- input: something like “yes” or “no”

```
Output robey.dialog.states.ordinaryStates.QuestionAnsweringState.reactToQuestion (Interp
Output robey.dialog.states.ordinaryStates.QuestionAnsweringState.useMemoryOrFallback (I
Output robey.dialog.states.ordinaryStates.QuestionAnsweringState.answerFromMemory (List
String robey.dialog.states.ordinaryStates.QuestionAnsweringState.inferMemoryAnswer (List
boolean robey.dialog.states.ordinaryStates.QuestionAnsweringState.userWantsGameCheck (I
boolean robey.dialog.states.ordinaryStates.QuestionAnsweringState.isIntentsHistoryComp
```

Private Members

```
final Logger robey.dialog.states.ordinaryStates.QuestionAnsweringState.logger = LogManager
int robey.dialog.states.ordinaryStates.QuestionAnsweringState.questionsAnswered = 0
boolean robey.dialog.states.ordinaryStates.QuestionAnsweringState.askingSpecifyingQues
String robey.dialog.states.ordinaryStates.QuestionAnsweringState.answerAfterUnspecifie
boolean robey.dialog.states.ordinaryStates.QuestionAnsweringState.userWantsGame = false
boolean robey.dialog.states.ordinaryStates.QuestionAnsweringState.robeySuggestedGame = f
```


Private Static Attributes

```
final String roboy.dialog.states.ordinaryStates.QuestionAnsweringState.TRANSITION_FINAL
final String roboy.dialog.states.ordinaryStates.QuestionAnsweringState.TRANSITION_LOOP
final String roboy.dialog.states.ordinaryStates.QuestionAnsweringState.TRANSITION_LOOP
final String roboy.dialog.states.ordinaryStates.QuestionAnsweringState.TRANSITION_TO_G
final int roboy.dialog.states.ordinaryStates.QuestionAnsweringState.MAX_NUM_OF_QUESTION
final RandomList<String> roboy.dialog.states.ordinaryStates.QuestionAnsweringState.rees
final RandomList<String> roboy.dialog.states.ordinaryStates.QuestionAnsweringState.ans
final double roboy.dialog.states.ordinaryStates.QuestionAnsweringState.THRESHOLD_BORED
```

```
template <T>
class RandomList : public java::util::ArrayList<T>
    Extension of ArrayList with possibility to get a random element.
```

Parameters

- <T>: Class of objects in this list

Public Functions

```
roboy.util.RandomList< T >.RandomList ()
```

Creates an empty ArrayList.

```
roboy.util.RandomList< T >.RandomList (T... objects)
```

Creates a list of objects that allows to select one element at random.

To prevent issues with heap pollution, use this constructor to reduce syntactic overhead only: `new RandomList("a", "b", "c")`

Parameters

- objects: objects to put in the list

```
roboy.util.RandomList< T >.RandomList (List< T > objectList)
```

Creates list of objects that allows to select one element at random.

Parameters

- objectList: list containing the objects to add to this list

```
T roboy.util.RandomList< T >.getRandomElement ()
```

Returns a random element from this list.

Return random element from this list

```
enum RelationshipAvailability
```

Public Members

```
roboy.memory.nodes.Interlocutor.RelationshipAvailability.ALL_AVAILABLE
roboy.memory.nodes.Interlocutor.RelationshipAvailability.SOME_AVAILABLE
roboy.memory.nodes.Interlocutor.RelationshipAvailability.NONE_AVAILABLE
```

class

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

Public Functions

```
roboy.memory.nodes.Roboy.Roboy (Neo4jMemoryInterface memory, String name)
```

Initializer for the *Roboy* node.

```
roboy.memory.nodes.Roboy.Roboy (Neo4jMemoryInterface memory)
```

Default 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 (Neo4jRelationship relation)
```

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 (Neo4jRelationship relationship, String name)
```

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

Public Static Attributes

```
final RandomList<Neo4jRelationship> roboy.memory.nodes.Roboy.VALID_NEO4J_RELATIONSHIPS
```

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

enum RoboyAbility

Implementations of Roboy's abilities.

Following methods have to be implemented:

- *wouldYouLikeToSeeDemoQuestions()* - provides a list of yes/no questions for the act method
- *demonstrateAbility()* - implementation of the ability (should block until demonstration is finished)
- *afterDemoEndedPhrases()* - provides a list of phrases that wrap up the ability demonstration

Public Functions

abstract RandomList<String> roboy.dialog.states.expoStates.RoboyAbility.wouldYouLikeTo

List of phrases for the act method.

Every phrase should ask the interlocutor whether he wants to see the ability in action.

Return list of phrases for the act method

abstract void roboy.dialog.states.expoStates.RoboyAbility.demonstrateAbility (RosMainNo

Implementation of the ability.

This method should block until the ability demonstration is finished.

abstract RandomList<String> roboy.dialog.states.expoStates.RoboyAbility.afterDemoEnded

List of phrases that wrap up the ability demonstration.

One of these phrases will be used after the demonstration is finished.

Return list of phrases that wrap up the ability demonstration

Public Members

roboy.dialog.states.expoStates.RoboyAbility.wouldYouLikeToSeeDemoQuestions

Private Static Attributes

static final Logger roboy.dialog.states.expoStates.RoboyAbility.logger = LogManager.getLog

enum RoboyEmotion

Comprises the emotions Roboy can demonstrate.

Public Functions

roboy.emotions.RoboyEmotion.RoboyEmotion (String type)

Public Members

roboy.emotions.RoboyEmotion.SHY =("shy")

roboy.emotions.RoboyEmotion.SMILE_BLINK =("smileblink")

roboy.emotions.RoboyEmotion.LOOK_LEFT =("lookleft")

roboy.emotions.RoboyEmotion.LOOK_RIGHT =("lookright")

roboy.emotions.RoboyEmotion.CAT_EYES =("catiris")

roboy.emotions.RoboyEmotion.KISS =("kiss")

roboy.emotions.RoboyEmotion.FACEBOOK_EYES =("img:facebook")

roboy.emotions.RoboyEmotion.NEUTRAL =("neutral")

String roboy.emotions.RoboyEmotion.type

class

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

Public Functions

robpy.io.RoboyNameDetectionInput.RoboyNameDetectionInput()
constructor which initialises recognition

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

Input robpy.io.RoboyNameDetectionInput.listen()
tracks what was said

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

Protected Attributes

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

class

Roboy Question Answering State.

This state will:

- offer the interlocutor to ask a question about Roboy
- retrieve the semantic parser result
- try to infer an asked question
- retrieve the relevant information from the Roboy node
- compose an answer
- fall back in case of failure

ExpoIntroductionState interface: 1) Fallback is required. 2) Outgoing transitions that have to be defined, following state if the question was answered:

- skills,
- abilities,
- newPerson. 3) Used 'infoFile' parameter containing Roboy answer phrases. Requires a path to RoboyInfoList.json

Public Functions

```
roboy.dialog.states.expoStates.RoboyQASState.RoboyQASState(String stateIdentifier, State)
Output roboy.dialog.states.expoStates.RoboyQASState.act()
Output roboy.dialog.states.expoStates.RoboyQASState.react(Interpretation input)
State roboy.dialog.states.expoStates.RoboyQASState.getNextState()
```

Public Static Attributes

```
final String roboy.dialog.states.expoStates.RoboyQASState.INTENTS_HISTORY_ID = "RQA"
```

Protected Functions

```
Set<String> roboy.dialog.states.expoStates.RoboyQASState.getRequiredTransitionNames()
```

Private Functions

```
String roboy.dialog.states.expoStates.RoboyQASState.inferMemoryAnswer(Interpretation input)
Neo4jRelationship roboy.dialog.states.expoStates.RoboyQASState.inferPredicateFromObject(Object)
String roboy.dialog.states.expoStates.RoboyQASState.extractNodeNameForPredicate(Neo4jRelationship)
```

Private Members

```
final String [] roboy.dialog.states.expoStates.RoboyQASState.TRANSITION_NAMES = { "skills", "a
final String [] roboy.dialog.states.expoStates.RoboyQASState.INTENT_NAMES = TRANSITION_NAMES
final String roboy.dialog.states.expoStates.RoboyQASState.INFO_FILE_PARAMETER_ID = "infoFile
final RandomList<String> roboy.dialog.states.expoStates.RoboyQASState.connectingPhrases
final RandomList<String> roboy.dialog.states.expoStates.RoboyQASState.roboyIntentPhrases
final Logger roboy.dialog.states.expoStates.RoboyQASState.LOGGER = LogManager.getLogger()
QAJsonParser roboy.dialog.states.expoStates.RoboyQASState.infoValues
State roboy.dialog.states.expoStates.RoboyQASState.nextState
boolean roboy.dialog.states.expoStates.RoboyQASState.intentIsFriend = false
```

enum RoboySkillIntent

Enum implementation of Roboy's skills.

General functionality:

- getRequestPhrase() - provides a phrase to offer some skills activity
- getResponsePhrase() - provides Roboy's response to the input
- getNegativeSentence() - provides response in case the intent was not POSITIVE

Specific functionality:

- getRandomJoke() - returns a string with a random joke

- getRandomFact() - returns a string with a random fact
- getAnswerFromSemanticParser() - tries to resolve the question with the semantic parser, returns the resulting string on success, uses generative model on failure

Public Functions

```
roboy.dialog.states.expoStates.RoboySkillIntent.RoboySkillIntent (String type)
String roboy.dialog.states.expoStates.RoboySkillIntent.getRequestPhrase ()
String roboy.dialog.states.expoStates.RoboySkillIntent.getResponsePhrase (Interpretation)
```

Public Members

```
roboy.dialog.states.expoStates.RoboySkillIntent.jokes = ("joke")
roboy.dialog.states.expoStates.RoboySkillIntent.fun_facts = ("fact")
roboy.dialog.states.expoStates.RoboySkillIntent.famous_entities = ("famous")
roboy.dialog.states.expoStates.RoboySkillIntent.math = ("math")
String roboy.dialog.states.expoStates.RoboySkillIntent.type
```

Private Functions

```
String roboy.dialog.states.expoStates.RoboySkillIntent.getRandomJoke (Linguistics.Utterance)
String roboy.dialog.states.expoStates.RoboySkillIntent.getRandomFact (Linguistics.Utterance)
String roboy.dialog.states.expoStates.RoboySkillIntent.getAnswerFromSemanticParser (Intent)
String roboy.dialog.states.expoStates.RoboySkillIntent.getNegativeSentence (String name)
```

Private Members

```
final Logger roboy.dialog.states.expoStates.RoboySkillIntent.LOGGER = LogManager.getLogger()
final RandomList<String> roboy.dialog.states.expoStates.RoboySkillIntent.connectingPhrases = ...
final RandomList<String> roboy.dialog.states.expoStates.RoboySkillIntent.negativePhrases = ...
final RandomList<String> roboy.dialog.states.expoStates.RoboySkillIntent.offerJokes = ...
final RandomList<String> roboy.dialog.states.expoStates.RoboySkillIntent.offerFacts = ...
final RandomList<String> roboy.dialog.states.expoStates.RoboySkillIntent.offerFamousEntities = ...
final RandomList<String> roboy.dialog.states.expoStates.RoboySkillIntent.offerMath = ...
final RandomList<String> roboy.dialog.states.expoStates.RoboySkillIntent.jokesList = ...
final RandomList<String> roboy.dialog.states.expoStates.RoboySkillIntent.factsList = ...
final RandomList<String> roboy.dialog.states.expoStates.RoboySkillIntent.parserError = ...
```

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")
class RosMainNode : public AbstractNodeMain
```

Public Functions

```
roboy.ros.RosMainNode.RosMainNode()
GraphName roboy.ros.RosMainNode.getDefaultNodeName()
void roboy.ros.RosMainNode.onStart(final ConnectedNode connectedNode)
void roboy.ros.RosMainNode.PerformMovement(String bodyPart, String name)
boolean roboy.ros.RosMainNode.SynthesizeSpeech(String text)
String roboy.ros.RosMainNode.RecognizeSpeech()
String roboy.ros.RosMainNode.GenerateAnswer(String question)
boolean roboy.ros.RosMainNode.ShowEmotion(RoboyEmotion emotion)
boolean roboy.ros.RosMainNode.ShowEmotion(String emotion)
String roboy.ros.RosMainNode.CreateMemoryQuery(String query)
String roboy.ros.RosMainNode.UpdateMemoryQuery(String query)
String roboy.ros.RosMainNode.GetMemoryQuery(String query)
String roboy.ros.RosMainNode.DeleteMemoryQuery(String query)
String roboy.ros.RosMainNode.CypherMemoryQuery(String query)
Object roboy.ros.RosMainNode.DetectIntent(String sentence)
void roboy.ros.RosMainNode.addListener(MessageListener listener, RosSubscribers subscri
boolean roboy.ros.RosMainNode.ApplyFilter(String filterName)
```

Protected Attributes

```
Object roboy.ros.RosMainNode.resp
```

Package Attributes

```
String roboy.ros.RosMainNode.warning = "Trying to talk to ROS package %s, but it's not initialized or deactivated"  
    "status : \"FAIL\", \" + \"message : \"Memory client not initialized.\"\" + \"}\" ]  
final Logger roboy.ros.RosMainNode.LOGGER = LogManager.getLogger()
```

Private Functions

```
void roboy.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.services = 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(RosServiceClients c)  
    Should always be called before getService, such that if a client failed to initialize, a fallback response can  
    be created instead.
```

Important if SHUTDOWN_ON_ROS_FAILURE is false.

```
boolean roboy.ros.RosManager.notInitialized(RosSubscribers s)
```

```
ServiceClient roboy.ros.RosManager.getService(RosServiceClients c)  
    Returns the ServiceClient matching the RosServiceClients entry.
```

the return might need casting before further use.

```
Subscriber roboy.ros.RosManager.getSubscriber(RosSubscribers s)
```

Package Attributes

```
final Logger roboy.ros.RosManager.LOGGER = LogManager.getLogger()
```


Private Functions

```
boolean robby.ros.RosManager.isAMemoryModule(String client)
```

Private Members

```
HashMap<RosServiceClients, ServiceClient> robby.ros.RosManager.serviceMap
```

```
HashMap<RosSubscribers, Subscriber> robby.ros.RosManager.subscriberMap
```

```
enum RosServiceClients
```

Stores the different client addresses and corresponding ROS message types.

Public Functions

```
robby.ros.RosServiceClients.RosServiceClients(String rosPackage, String address, String type)
```

Public Members

```
robby.ros.RosServiceClients.SPEECHSYNTHESIS = ("robby_speech_synthesis", "/robby/cognition/speech/synthesis", DetectFace._TYPE)
```

```
robby.ros.RosServiceClients.GENERATIVE = ("robby_gnlp", "/robby/cognition/generative_nlp/answer", GenerateAnswer._TYPE)
```

```
robby.ros.RosServiceClients.FACEDETECTION = ("robby_vision", "/speech_synthesis/talk", DetectFace._TYPE)
```

```
robby.ros.RosServiceClients.OBJECTRECOGNITION = ("robby_vision", "/speech_synthesis/talk", RecognizeObject._TYPE)
```

```
robby.ros.RosServiceClients.STT = ("robby_speech_recognition", "/robby/cognition/speech/recognition", RecognizeSpeech._TYPE)
```

```
robby.ros.RosServiceClients.EMOTION = ("robby_face", "/robby/cognition/face/emotion", ShowEmotion._TYPE)
```

```
robby.ros.RosServiceClients.CREATEMEMORY = ("robby_memory", "/robby/cognition/memory/create", DataQuery._TYPE)
```

```
robby.ros.RosServiceClients.UPDATEMEMORY = ("robby_memory", "/robby/cognition/memory/update", DataQuery._TYPE)
```

```
robby.ros.RosServiceClients.GETMEMORY = ("robby_memory", "/robby/cognition/memory/get", DataQuery._TYPE)
```

```
robby.ros.RosServiceClients.DELETEMEMORY = ("robby_memory", "/robby/cognition/memory/remove", DataQuery._TYPE)
```

```
robby.ros.RosServiceClients.CYPHERMEMORY = ("robby_memory", "/robby/cognition/memory/cypher", DataQuery._TYPE)
```

```
robby.ros.RosServiceClients.INTENT = ("robby_intents", "/robby/cognition/detect_intent", DetectIntent._TYPE)
```

```
robby.ros.RosServiceClients.SNAPCHATFILTER = ("robby_filters", "/robby/cognition/apply_filter", ApplyFilter._TYPE)
```

```
String robby.ros.RosServiceClients.rosPackage
```

```
String robby.ros.RosServiceClients.address
```

```
String robby.ros.RosServiceClients.type
```

```
enum RosSubscribers
```

Public Functions

```
robby.ros.RosSubscribers.RosSubscribers(String rosPackage, String address, String type)
```

Public Members

```
roboy.ros.RosSubscribers.DIRECTION_VECTOR = ("roboy_audio", "/roboy/cognition/audio/direction_of_arrival", DIRECTION_VECTOR)
roboy.ros.RosSubscribers.FACE_COORDINATES = ("roboy_vision", "/roboy/cognition/vision/coordinates", FaceCoordinates)
roboy.ros.RosSubscribers.NEW_FACIAL_FEATURES = ("roboy_vision", "/roboy/cognition/vision/features", NewFacialFeatures)
roboy.ros.RosSubscribers.TEST_TOPIC = ("roboy_test", "/roboy", std_msgs.String._TYPE)

String roboy.ros.RosSubscribers.rosPackage
String roboy.ros.RosSubscribers.address
String roboy.ros.RosSubscribers.type

class ROSTest : public roboy::context::ValueHistory<String>
    For testing a ROS topic connection which sends simple String messages.

class ROSTestUpdater : public roboy::context::ROSTopicUpdater<std_msgs.String, ROSTest>
    For testing a ROS topic connection which sends simple String messages.
```

Public Functions

```
roboy.context.contextObjects.ROSTestUpdater.ROSTestUpdater(ROSTest target, RosMainNode)
```

Protected Functions

```
RosSubscribers roboy.context.contextObjects.ROSTestUpdater.getTargetSubscriber()

synchronized void roboy.context.contextObjects.ROSTestUpdater.update()

template <Message, Target>
class
```

An external updater connected to a ROS topic that can push the arriving values to the target.

The update() method should implement the logic of adding to the target.

Parameters

- <Message>: Type of messages from the ROS topic.
- <Target>: The target object to be updated.

Subclassed by *roboy.context.contextObjects.AudioDirectionUpdater*, *roboy.context.contextObjects.ROSTestUpdater*

Public Functions

```
roboy.context.ROSTopicUpdater< Message, Target >.ROSTopicUpdater(Target target, RosMainNode)
```

Protected Functions

```
abstract RosSubscribers roboy.context.ROSTopicUpdater< Message, Target >.getTargetSubscriber()
    Implement this in the subclass to define the ROS subscriber this updater should use.
```

```
void roboy.context.ROSTopicUpdater< Message, Target >.addListener(MessageListener listener)
```

Protected Attributes

```
final Target roboy.context.ROSTopicUpdater< Message, Target >.target
volatile Message roboy.context.ROSTopicUpdater< Message, Target >.message
final RosSubscribers roboy.context.ROSTopicUpdater< Message, Target >.targetSubscriber
```

Private Functions

```
void roboy.context.ROSTopicUpdater< Message, Target >.start(RosMainNode ros)
    Starts a new MessageListener.
```

class Segue

A segue /swe/ is a smooth transition from one topic to the next.

(c) Wikipedia

Dialog states can decide to add segues to their output (in act() or react()) to improve the dialog flow. Segues are categorized by types and will be inserted into the conversation with a certain probability.

Special options:

- “%s” inside a segue will be replaced with interlocutor’s name if available. If no interlocutor is available all segues with “%s” won’t be used.

Public Functions

```
roboy.dialog.Segue.Segue(SegueType type)
    Creates a segue based on a type and sets the default appearance probability.
```

Parameters

- type: segue type

```
roboy.dialog.Segue.Segue(SegueType type, double probability)
    Creates a segue based on a type and specified appearance probability.
```

Note that

Parameters

- type: segue type
- probability: probability to use this segue in the conversation

```
SegueType roboy.dialog.Segue.getType()
double roboy.dialog.Segue.getProbability()
```

Public Static Attributes

```
final double roboy.dialog.Segue.DEFAULT_PROBABILITY = 0.3
    Default segue usage probability.
```

Private Members

```
final SegueType roboy.dialog.Segue.type
final double roboy.dialog.Segue.probability
```

enum SegueType

Definitions of segue types here.

Note: all segues are stored in this class for simplicity and may be moved into a separate file later.

Public Functions

```
abstract RandomList<String> roboy.dialog.Segue.SegueType.getPossibleSegues()
```

Returns a list of possible segues for this segue type as strings.

Return list of possible segues

Public Members

```
roboy.dialog.Segue.SegueType.getPossibleSegues
```

class

Semantic parser class.

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

Public Functions

```
roboy.linguistics.sentenceanalysis.SemanticParserAnalyzer.SemanticParserAnalyzer()
```

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.

Public Static Functions

```
static void roboy.linguistics.sentenceanalysis.SemanticParserAnalyzer.main(String[] ar
```

Testing function.

Private Functions

String `roboy.linguistics.sentenceanalysis.SemanticParserAnalyzer.get_answers`(String and
Function reading parser answer in returned JSON string.

List can contain triples, strings or doubles.

Return String formed by joined list.

Parameters

- `answer`: String containing parser answer received by analyzer.

List<Triple> `roboy.linguistics.sentenceanalysis.SemanticParserAnalyzer.extract_relations`
Function reading extracted relations in returned JSON string.

Return List of triple objects with relations extracted.

Parameters

- `relations`: Map of relations and their confidence.

List<Triple> `roboy.linguistics.sentenceanalysis.SemanticParserAnalyzer.extract_triples`
Function reading triples from returned JSON string.

Return List of triple objects with RDF triples extracted.

Parameters

- `input`: parsing result

Private Members

SemanticAnalyzerInterface `roboy.linguistics.sentenceanalysis.SemanticParserAnalyzer.sem`

Private Static Attributes

final Logger `roboy.linguistics.sentenceanalysis.SemanticParserAnalyzer.logger` = LogManager

enum `SemanticRole`

Public Members

`roboy.linguistics.Linguistics.SemanticRole.PREDICATE`

`roboy.linguistics.Linguistics.SemanticRole.AGENT`

`roboy.linguistics.Linguistics.SemanticRole.PATIENT`

`roboy.linguistics.Linguistics.SemanticRole.TIME`

`roboy.linguistics.Linguistics.SemanticRole.LOCATION`

`roboy.linguistics.Linguistics.SemanticRole.MANNER`

`roboy.linguistics.Linguistics.SemanticRole.INSTRUMENT`

`roboy.linguistics.Linguistics.SemanticRole.ORIGIN`

`roboy.linguistics.Linguistics.SemanticRole.DESTINATION`

```
roboy.linguistics.Linguistics.SemanticRole.RECIPIENT
roboy.linguistics.Linguistics.SemanticRole.BENEFICIARY
roboy.linguistics.Linguistics.SemanticRole.PURPOSE
roboy.linguistics.Linguistics.SemanticRole.CAUSE
```

class

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

Public Functions

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

Private Functions

```
Interpretation roboy.linguistics.sentenceanalysis.SentenceAnalyzer.extractPAS(String s)
Triple roboy.linguistics.sentenceanalysis.SentenceAnalyzer.analyzeStatement(List< String > t)
Triple roboy.linguistics.sentenceanalysis.SentenceAnalyzer.analyzeIsIt(List< String > t)
Triple roboy.linguistics.sentenceanalysis.SentenceAnalyzer.analyzeDoesIt(List< String > t)
Triple roboy.linguistics.sentenceanalysis.SentenceAnalyzer.analyzeWho(List< String > t)
Triple roboy.linguistics.sentenceanalysis.SentenceAnalyzer.analyzeWhat(List< String > t)
Triple roboy.linguistics.sentenceanalysis.SentenceAnalyzer.analyzeHowIs(List< String > t)
Triple roboy.linguistics.sentenceanalysis.SentenceAnalyzer.analyzeHowDo(List< String > t)
```

Private Members

```
Map<String, String> roboy.linguistics.sentenceanalysis.SentenceAnalyzer.meanings
enum SentenceType
```

Public Members

```
roboy.linguistics.Linguistics.SentenceType.GREETING
roboy.linguistics.Linguistics.SentenceType.FAREWELL
roboy.linguistics.Linguistics.SentenceType.SEGUE
roboy.linguistics.Linguistics.SentenceType.ANECDOTE
roboy.linguistics.Linguistics.SentenceType.STATEMENT
roboy.linguistics.Linguistics.SentenceType.NONE
roboy.linguistics.Linguistics.SentenceType.WHO
roboy.linguistics.Linguistics.SentenceType.HOW
```

```

roboy.linguistics.Linguistics.SentenceType.HOW_IS
roboy.linguistics.Linguistics.SentenceType.HOW_DO
roboy.linguistics.Linguistics.SentenceType.WHY
roboy.linguistics.Linguistics.SentenceType.WHEN
roboy.linguistics.Linguistics.SentenceType.WHERE
roboy.linguistics.Linguistics.SentenceType.WHAT
roboy.linguistics.Linguistics.SentenceType.IS_IT
roboy.linguistics.Linguistics.SentenceType.DOES_IT

```

class

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

This class was deprecated because the tokens are anyways overwritten by *SemanticParserAnalyzer*. *SemanticParserAnalyzer* is run next and does not require the analysis of *SimpleTokenizer* to function. Furthermore ST's approach to Parsing is not exactly correct.

[Roboy]: Do I need a visa to go to suzhou? [You]: Yes Roboy, you do

Simple Tokenizer: 0 = "yes" 1 = "robey," 2 = "you" 3 = "do"

Semantic Parser *Analyzer*: (Desired) 0 = "yes" 1 = "robey" 2 = "," 3 = "you" 4 = "do"

Public Functions

```

Interpretation robey.linguistics.sentenceanalysis.SimpleTokenizer.analyze(Interpretation

```

Private Functions

```

List<String> robey.linguistics.sentenceanalysis.SimpleTokenizer.tokenize(String sentence)

```

class

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 in use.

Public Functions

```

robey.linguistics.phonetics.SoundexEncoder.SoundexEncoder(Soundex soundex)
String robey.linguistics.phonetics.SoundexEncoder.encode(String input)

```

Private Members

```

Soundex robey.linguistics.phonetics.SoundexEncoder.soundex

```

class

Action used for talking.

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

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 *Output.useFallback()* 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.dialog.states.botboy.BotBoyState*, *roboy.dialog.states.definitions.ExpoState*, *roboy.dialog.states.eventStates.UzupisState*, *roboy.dialog.states.expoStates.DemonstrateAbilitiesState*, *roboy.dialog.states.gameStates.ChooseGameState*, *roboy.dialog.states.gameStates.GamingSnapchatState*, *roboy.dialog.states.gameStates.GamingTwentyQuestionsState*, *roboy.dialog.states.ordinaryStates.FarewellState*, *roboy.dialog.states.ordinaryStates.IntroductionState*, *roboy.dialog.states.ordinaryStates.PassiveGreetingsState*, *roboy.dialog.states.ordinaryStates.PersonalInformationAskingState*, *roboy.dialog.states.ordinaryStates.PersonalInformationFollowingState*, *roboy.dialog.states.ordinaryStates.QuestionAnsweringState*, *roboy.dialog.states.ordinaryStates.WildTalkState*, *roboy.dialog.tutorials.tutorialStates.DoYouKnowMathState*, *roboy.dialog.tutorials.tutorialStates.ToyFarewellState*, *roboy.dialog.tutorials.tutorialStates.ToyGreetingsState*, *roboy.dialog.tutorials.tutorialStates.ToyIntroState*, *roboy.dialog.tutorials.tutorialStates.ToyRandomAnswerState*

Public Functions

roboy.dialog.states.definitions.State.State(String stateIdentifier, StateParameters parameters)

Create a state object with given identifier (state name) and parameters.

The parameters should contain a reference to a state machine for later use. The state will not automatically add itself to the state machine.

Parameters

- `stateIdentifier`: identifier (name) of this state
- `params`: parameters for this state, should contain a reference to a state machine

String roboy.dialog.states.definitions.State.getIdentifier()

void roboy.dialog.states.definitions.State.setIdentifier(String stateIdentifier)

StateParameters `roboy.dialog.states.definitions.State.getParameters()`

final State `roboy.dialog.states.definitions.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.dialog.states.definitions.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.dialog.states.definitions.State.setTransition(String name, State goTo)`

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.dialog.states.definitions.State.getTransition(String name)`

final HashMap<String, State> `roboy.dialog.states.definitions.State.getAllTransitions()`

final void `roboy.dialog.states.definitions.State.setOptionalPersFileInfo(String key, S`

Set personality file additional information like state comment.

[!] Do not use it in your state code! This info is only stored to make sure we don't lose the comment etc. when saving this state to file.

final String `roboy.dialog.states.definitions.State.getOptionalPersFileInfo(String key)`

Get personality file additional information like state comment.

[!] Do not use it in your state code! This info is only stored to make sure we don't lose the comment etc. when saving this state to file.

abstract Output `roboy.dialog.states.definitions.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 Output `roboy.dialog.states.definitions.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 (should not be null)

Parameters

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

abstract State `roboy.dialog.states.definitions.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

boolean `roboy.dialog.states.definitions.State.isFallbackRequired()`

This function can be overridden to sub classes to indicate that this state can require a fallback.

If this is the case, but no fallback was defined, you will be warned.

Return true if this state requires a fallback and false otherwise

final boolean `roboy.dialog.states.definitions.State.allRequiredTransitionsAreInitialized()`

Checks if all required transitions were initialized correctly.

Required transitions are defined in [*getRequiredTransitionNames\(\)*](#).

Return true if all required transitions of this state were initialized correctly

final boolean `roboy.dialog.states.definitions.State.allRequiredParametersAreInitialized()`

Checks if all required parameters were initialized correctly.

Required parameters are defined in [*getRequiredParameterNames\(\)*](#).

Return true if all required parameters of this state were initialized correctly

JsonObject `roboy.dialog.states.definitions.State.toJsonObject()`

Create a JSON representation for this state.

Only the identifier, class name, transitions, parameters and fallback identifier are saved. Internal other internal variables are ignored.

Return JSON representation for this state

String `roboy.dialog.states.definitions.State.toString()`

boolean `roboy.dialog.states.definitions.State.equals(Object obj)`

Protected Functions

Set<String> `roboy.dialog.states.definitions.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.dialog.states.definitions.State.getRequiredParameterNames()`

Set<String> `roboy.dialog.states.definitions.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

DialogStateMachine `robey.dialog.states.definitions.State.getStateMachine()`
 Shortcut for `getParameters().getStateMachine()`

Return *DialogStateMachine*

RosMainNode `robey.dialog.states.definitions.State.getRosMainNode()`
 Shortcut for `getParameters().getRosMainNode()`

Return `RosMainNode` (if previously provided to the *DialogStateMachine*)

Neo4jMemoryInterface `robey.dialog.states.definitions.State.getMemory()`
 Shortcut for `getParameters().getMemory()`

Return `Neo4jMemoryInterface` (if previously provided to the *DialogStateMachine*)

InferenceEngine `robey.dialog.states.definitions.State.getInference()`
 Shortcut for `getParameters().getInference()`

Return `InferenceEngine`

Context `robey.dialog.states.definitions.State.getContext()`
 Shortcut for `getParameters().getStateMachine().getContext()`

Return `Context`

RandomList<MemoryNodeModel> `robey.dialog.states.definitions.State.getMemNodesByIds(Array)`
 Helper function Ask memory to return nodes for given ids.

Return Collection of `MemoryNodeModels`

Parameters

- `ids`: ids for memory

Private Functions

boolean `robey.dialog.states.definitions.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

final Logger `robey.dialog.states.definitions.State.logger` = `LogManager.getLogger()`

String `robey.dialog.states.definitions.State.stateIdentifier`

StateParameters `robey.dialog.states.definitions.State.parameters`

State `robey.dialog.states.definitions.State.fallback`

HashMap<String, State> `robey.dialog.states.definitions.State.transitions`

HashMap<String, String> roboy.dialog.states.definitions.State.optionalPersFileInfo
Personality file additional information: everything like state comment goes here.

[!!] Do not use it in your state code! This info is only stored to make sure we don't lose the comment etc. when saving this state to file.

class

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.

Public Functions

roboy.dialog.personality.StateBasedPersonality.StateBasedPersonality(InferenceEngine i

void roboy.dialog.personality.StateBasedPersonality.reset()

Reset this state machine: active state will be set to initial state.

All State objects will stay as they are and will KEEP all internal variables unchanged.

If you do not want to keep any information from the previous conversation, call *loadFromFile()* before reset. Reloading from file will create "fresh" State objects.

boolean roboy.dialog.personality.StateBasedPersonality.conversationEnded()

Indicates that the conversation should stop.

This happens if

- the active state returns no next state
- or the active state returns END_CONVERSATION from act() or react()

Return true if the conversation is finished and this personality should be reset or reloaded from file.

List<Action> roboy.dialog.personality.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.dialog.personality.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.dialog.personality.StateBasedPersonality.endConversation()

Internal function, cleanup before the conversation ends:

- set active state to null
- set stopTalking flag to true

void roboy.dialog.personality.StateBasedPersonality.stateAct(State state, List< Action>

Call the act function of the state, verbalize the interpretation (if any) and add it to the list of actions.

Parameters

- state: state to call ACT on
- previousActions: list of previous action to append the verbalized result

void roboy.dialog.personality.StateBasedPersonality.stateReact(State state, Interpretation

Call the react function of the state.

If the state can't react, recursively ask fallbacks. Verbalize the resulting reaction and add it to the list of actions.

Parameters

- state: state to call REact on
- input: input from the person Roboy speaks to
- previousActions: list of previous action to append the verbalized result

void roboy.dialog.personality.StateBasedPersonality.segueHandler(List< Action > prev

Decide whether to use segue or not and append it to the end of the list of previous actions.

Parameters

- previousActions: list of previous actions
- segue: segue object

void roboy.dialog.personality.StateBasedPersonality.exceptionHandler(State state, Except

Internal function, handles exceptions coming from act() or react(...) functions.

In general, act() and react(...) should never throw an exception unless the implementation is buggy. In that case the dialog system will log the error and try to continue working.

Parameters

- state: state that threw the exception
- e: exception
- previousActions: list of previous planned actions, will be extended
- comesFromAct: indicates whether act() or react(...) threw the exception

Private Members

final Logger roboy.dialog.personality.StateBasedPersonality.logger = LogManager.getLogger()

final Verbalizer roboy.dialog.personality.StateBasedPersonality.verbalizer

```
boolean roboy.dialog.personality.StateBasedPersonality.stopTalking
```

class StateFactory

This class is used to create *State* objects based on the class name (as a string).

Use case: The personality file defines an implementation for each state. The implementation is a simple string that contains the class name.

Example: Using some magic, this class would convert a string into a proper Java object: roboy.dialog... ToyGreetingsState” > {Java object of class ToyGreetingsState}

Important note for the state implementation: The magic that is used here is called Java Reflection. It adds some small restriction to the implementation of the states: Every *State* sub-class must have a constructor that takes exactly two parameters. The first one is a String, the second is an object of type *StateParameters*. For example: ToyGreetingsState(String id, StateParameters params)

You can have other constructors as well.

Public Static Functions

```
static State roboy.dialog.states.definitions.StateFactory.createStateByClassName (String
```

Create a Java *State* object based on the provided class name.

Full class name must be used: ‘my.package.asdf.StateName’ instead of ‘StateName’. The class must be a sub-class of *State*.

This function doesn’t throw Exceptions and will return null if something goes wrong.

Return a new instance of a *State* object of specified class OR null if something goes wrong

Parameters

- className: class name of the *State* object to be created
- stateIdentifier: state identifier/name (this is NOT the class name!)
- parameters: state parameters

Private Static Attributes

```
final Logger roboy.dialog.states.definitions.StateFactory.logger = LogManager.getLogger()
```

class StateFactoryTest

Tests related to the StateFactory.

Public Functions

```
void roboy.dialog.StateFactoryTest.factoryCreatesCorrectStateObjects ()
```

```
void roboy.dialog.StateFactoryTest.factoryDoesNotBreakOnInvalidClassNames ()
```

class StateMachineEqualityTest

Tests related to state machine equality.

Public Functions

```

void robey.dialog.StateMachineEqualityTest.machineEqualsItself()
void robey.dialog.StateMachineEqualityTest.stringEqualsCode()
void robey.dialog.StateMachineEqualityTest.notEqualsNoInitialState()
void robey.dialog.StateMachineEqualityTest.notEqualsDifferentStates()
void robey.dialog.StateMachineEqualityTest.notEqualsDifferentTransitions()

```

class StateMachineExamples

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

Public Static Functions

```

static void robey.dialog.tutorials.StateMachineExamples.main(String[] args)

```

Private Static Functions

```

static DialogStateMachine robey.dialog.tutorials.StateMachineExamples.fromCode()
static DialogStateMachine robey.dialog.tutorials.StateMachineExamples.fromFile()
static DialogStateMachine robey.dialog.tutorials.StateMachineExamples.fromString()

```

Private Static Attributes

```

final String robey.dialog.tutorials.StateMachineExamples.toyPersonality

```

class StateMachineInitializationTest

Tests related to the state machine initialization and loading from file/string.

Public Functions

```

void robey.dialog.StateMachineInitializationTest.activeStateIsSetToInitialState()
void robey.dialog.StateMachineInitializationTest.machineContainsAllStates()
void robey.dialog.StateMachineInitializationTest.transitionsAreOK()
void robey.dialog.StateMachineInitializationTest.parametersAreOK()
void robey.dialog.StateMachineInitializationTest.fallbackIsOK()

```

class StatementBuilder

Public Static Functions

```

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

Return

Parameters

- input:
- list:

class StateParameters

An object containing all parameters that can be interesting for an arbitrary *State* object.

String parameters are defined in the personality file. Other important parameters are references to the *DialogStateMachine* and the RosMainNode.

Public Functions

```
roboy.dialog.states.definitions.StateParameters.StateParameters (DialogStateMachine sta  
roboy.dialog.states.definitions.StateParameters.StateParameters (DialogStateMachine sta  
StateParameters roboy.dialog.states.definitions.StateParameters.setParameter (String pa  
String roboy.dialog.states.definitions.StateParameters.getParameter (String parameterNa  
HashMap<String, String> roboy.dialog.states.definitions.StateParameters.getAllParamete  
DialogStateMachine roboy.dialog.states.definitions.StateParameters.getStateMachine ()  
RosMainNode roboy.dialog.states.definitions.StateParameters.getRosMainNode ()  
Neo4jMemoryInterface roboy.dialog.states.definitions.StateParameters.getMemory ()  
InferenceEngine roboy.dialog.states.definitions.StateParameters.getInference ()
```

Private Members

```
final Logger roboy.dialog.states.definitions.StateParameters.logger = LogManager.getLogger()  
final HashMap<String, String> roboy.dialog.states.definitions.StateParameters.paramNam  
final DialogStateMachine roboy.dialog.states.definitions.StateParameters.stateMachine  
final RosMainNode roboy.dialog.states.definitions.StateParameters.rosMainNode  
final Neo4jMemoryInterface roboy.dialog.states.definitions.StateParameters.memory  
final InferenceEngine roboy.dialog.states.definitions.StateParameters.inference
```

class

Singleton Class For Telegram Bot.

Public Functions

void roboy.util.TelegramCommunicationHandler.onUpdateReceived(Update update)
 Receives the updates from telegram's api and called by it.

Parameters

- update: consist of an update of a chat.

void roboy.util.TelegramCommunicationHandler.onTimeout(String chatID)
 Called when specified time passed w.r.t.

unique chatID. Concatenate all the messages that is not processed. Calls the InputDevice for telegram

Parameters

- chatID: unique identifier for a chat.

void roboy.util.TelegramCommunicationHandler.sendMessage(String message, String chatID)
 Called from the OutputDevice when a message desired to send Initiates the "typing status" and waits for a specified time Sends the message afterwards.

Parameters

- chatID: unique identifier for a chat.

void roboy.util.TelegramCommunicationHandler.sendSticker(String chatID, String sticker)
 Called from the OutputDevice when a message desired to send with stickers Directly sends the sticker, without waiting for a specified time!

Parameters

- chatID: unique identifier for a chat.
- stickerId: unique identifier for a sticker.

void roboy.util.TelegramCommunicationHandler.sendTypingFromChatID(String chatID)
 Sends the "typing" status to the telegram chat.

Parameters

- chatID: unique identifier for a chat.

void roboy.util.TelegramCommunicationHandler.sendImageFromUrl(String url, String chatID)
 Sends a image from url to the desired chat.

Parameters

- url: image's url
- chatId: unique identifier for a chat

void roboy.util.TelegramCommunicationHandler.sendImageFromFileId(String fileId, String chatID)
 Sends a image from fileId to the desired chat.

Parameters

- fileId: a file id that is produced by telegram and using by it
- chatId: unique identifier for a chat

void roboy.util.TelegramCommunicationHandler.sendImageUploadingAFile(String filePath, String chatID)
 Sends a file to the desired chat.

Parameters

- `filePath`: path of a file
- `chatId`: unique identifier for a chat

```
String roboy.util.TelegramCommunicationHandler.getBotUsername()
```

```
String roboy.util.TelegramCommunicationHandler.getBotToken()
```

Public Static Functions

```
static TelegramCommunicationHandler roboy.util.TelegramCommunicationHandler.getInstance()
```

Private Functions

```
roboy.util.TelegramCommunicationHandler.TelegramCommunicationHandler()
```

```
void roboy.util.TelegramCommunicationHandler.handleTimeout(String chatID)
```

Waits until specified time passed after the last message w.r.t.

given chatId.

Parameters

- `chatID`: unique identifier for a chat.

```
String roboy.util.TelegramCommunicationHandler.getJsonString(String key)
```

Private Members

```
volatile List<Pair<String, String> > roboy.util.TelegramCommunicationHandler.pairs = new
```

```
List<Timeout> roboy.util.TelegramCommunicationHandler.telegramTimeouts
```

Private Static Attributes

```
final Logger roboy.util.TelegramCommunicationHandler.logger = LogManager.getLogger()
```

```
final String roboy.util.TelegramCommunicationHandler.tokensPath = ConfigManager.TELEGRAM_A
```

```
final int roboy.util.TelegramCommunicationHandler.TYPING_TIME_LIMIT = 3
```

```
final int roboy.util.TelegramCommunicationHandler.INPUT_TIME_LIMIT = 5
```

```
final int roboy.util.TelegramCommunicationHandler.initTime = (int) (System.currentTimeMillis() / 100
```

```
TelegramCommunicationHandler roboy.util.TelegramCommunicationHandler.instance
```

class

Handles telegram API and hands threads their respective messages.

Public Functions

roboy.io.TelegramInput.TelegramInput (String uuid)

Creates a Telegraminput device that sorts incoming messages from the telegram handle to the individual conversations.

Parameters

- **uuid**: The uuid of the interlocutor must be formed like this: “telegram-[uuid from service]”

Input roboy.io.TelegramInput.listen ()

Thread waits in *listen()* until a new input is provided and the thread is interrupted, then returns with said input.

If the thread is interrupted without *Input* waiting to be consumed, *listen()* throws an IOException

Exceptions

- **InterruptedException**: InterruptedException thrown by the thread when interrupted while wait()ing

void roboy.io.TelegramInput.cleanup ()

Deregisters the instance from the static ledger.

Must be called when it should be destroyed or it will stay in memory until the end of operation.

Public Static Functions

static void roboy.io.TelegramInput.onUpdate (Pair < String, String > update)

Gets called by the TelegramAPI Thread whenever a new telegram message arrives.

Places them in the appropriate thread’s message string. Creates thread beforehand, if necessary.

Parameters

- **update**: contains a (sender uuid,message) string pair.

Private Members

volatile String roboy.io.TelegramInput.message

Private Static Attributes

final Logger roboy.io.TelegramInput.logger = LogManager.getLogger()

final HashMap<String, TelegramInput> roboy.io.TelegramInput.inputDevices = new HashMap<>()

class

Public Functions

roboy.io.TelegramOutput.TelegramOutput (String uuid)

Handles sending messages to the TelegramAPI from the Dialog System.

Parameters

- `uuid`: The uuid of the interlocutor must be formed like this: “telegram-[uuid from service]”

`void robey.io.TelegramOutput.act(List< Action > actions)`

Carries out actions in the telegram way: Speechactions are sent as text messages via telegram, Emotion-Actions are sent as stickers via telegram.

Parameters

- `actions`: Actions to be carried out on the telegram service

Private Members

`TelegramCommunicationHandler robey.io.TelegramOutput.communicationHandler = TelegramCommuni`

`String robey.io.TelegramOutput.uuid`

Private Static Attributes

`final Logger robey.io.TelegramOutput.logger = LogManager.getLogger()`

`class Term`

Public Functions

`List<String> robey.linguistics.Term.getPos()`

`void robey.linguistics.Term.setPos(List< String > pos)`

`float robey.linguistics.Term.getProbability()`

`void robey.linguistics.Term.setProbability(float prob)`

`String robey.linguistics.Term.getConcept()`

`void robey.linguistics.Term.setConcept(String concept)`

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

`boolean robey.linguistics.Term.equals(Object obj)`

`int robey.linguistics.Term.hashCode()`

Private Members

`List<String> robey.linguistics.Term.pos = null`

`float robey.linguistics.Term.probability = 0`

`String robey.linguistics.Term.concept = null`

`class Timeout`

Public Functions

```

roboy.util.Timeout.Timeout(long millis)
void roboy.util.Timeout.setMillis(long millis)
long roboy.util.Timeout.getMillis()
String roboy.util.Timeout.getUnique()
void roboy.util.Timeout.setUnique(String unique)
void roboy.util.Timeout.start(TimeoutObserver timeoutObserver)
void roboy.util.Timeout.stop()

```

Private Members

```

Runnable roboy.util.Timeout.runnable
long roboy.util.Timeout.millis
Timer roboy.util.Timeout.timer
TimerTask roboy.util.Timeout.timerTask
String roboy.util.Timeout.unique

```

interface TimeoutObserver

Subclassed by *roboy.util.TelegramCommunicationHandler*

Public Functions

```
void roboy.util.Timeout.TimeoutObserver.onTimeout(String unique)
```

template <V>

class

Sample implementation of a *ValueHistory* using timestamps (longs) as keys and a TreeMap for data storage.

The timestamps are equal or larger than the time when *updateValue()* was called. Implementation does not guarantee perfect timestamp accuracy, but achieves key uniqueness.

Public Functions

```
roboy.context.TimestampedValueHistory< V >.TimestampedValueHistory()
```

```
synchronized V roboy.context.TimestampedValueHistory< V >.getValue()      Return
```

The last element added to this history, or null if not found.

```
synchronized TreeMap<Long, V> roboy.context.TimestampedValueHistory< V >.getLastNValue
```

Get a copy of the last n entries added to the history.

Less values may be returned if there are not enough values in this history. In case of no values, an empty map is returned.

Needs to be synchronized because data cannot be changed while working with an Iterator.

```
synchronized void roboy.context.TimestampedValueHistory< V >.updateValue(V value)
```

Puts a value into the history in the last place.

```
int roboy.context.TimestampedValueHistory< V >.getNumberOfValuesSinceStart()  
synchronized boolean roboy.context.TimestampedValueHistory< V >.contains(V value)  
synchronized boolean roboy.context.TimestampedValueHistory< V >.purgeHistory()
```

Private Functions

```
synchronized long roboy.context.TimestampedValueHistory< V >.generateKey()
```

Private Members

```
volatile long roboy.context.TimestampedValueHistory< V >.lastTime  
    Marks the last time a value was added to the history (or initialization).  
TreeMap<Long, V> roboy.context.TimestampedValueHistory< V >.data  
int roboy.context.TimestampedValueHistory< V >.totalValuesAdded
```

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 always acts with “Bye bye” to indicate the end of conversation.

The interlocutor’s answer is ignored and there is no reaction (*Output.sayNothing()*). This ends the conversation (returning null in *getNextState()*).

ToyFarewellState interface: 1) Fallback is not required. 2) This state has no outgoing transitions. 3) No parameters are used.

Public Functions

```
roboy.dialog.tutorials.tutorialStates.ToyFarewellState.ToyFarewellState(String stateId)
Output roboy.dialog.tutorials.tutorialStates.ToyFarewellState.act()
Output roboy.dialog.tutorials.tutorialStates.ToyFarewellState.react(Interpretation input)
State roboy.dialog.tutorials.tutorialStates.ToyFarewellState.getNextState()
```

class

This is a very simple example how you could implement an initial state.

Roboy will greet the interlocutor (the guy he speaks to) 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.

ToyGreetingsState interface: 1) Fallback is required. 2) Outgoing transitions that have to be defined:

- next: following state if there was a greeting
- noHello: following state if there was NO greeting 3) No parameters are used.

Public Functions

```
roboy.dialog.tutorials.tutorialStates.ToyGreetingsState.ToyGreetingsState(String stateId)
Output roboy.dialog.tutorials.tutorialStates.ToyGreetingsState.act()
Output roboy.dialog.tutorials.tutorialStates.ToyGreetingsState.react(Interpretation input)
State roboy.dialog.tutorials.tutorialStates.ToyGreetingsState.getNextState()
boolean roboy.dialog.tutorials.tutorialStates.ToyGreetingsState.isFallbackRequired()
```

Protected Functions

```
Set<String> roboy.dialog.tutorials.tutorialStates.ToyGreetingsState.getRequiredTransitions()
```

Private Members

```
State roboy.dialog.tutorials.tutorialStates.ToyGreetingsState.next
```

class

ToyIntroState demonstrates a simple introduction.

A single parameter is used. Roboy will tell the interlocutor his name and ask for the Interlocutor’s name. The reply is ignored to keep this example simple.

ToyIntroState interface: 1) Fallback is not required. 2) Outgoing transitions that have to be defined:

- next: following state 3) Names of the parameters that have to be passed to the constructor:
- introductionSentence: A sentence that should be used as introduction

Public Functions

```
roboy.dialog.tutorials.tutorialStates.ToyIntroState.ToyIntroState(String stateIdentifi
Output roboy.dialog.tutorials.tutorialStates.ToyIntroState.act ()
Output roboy.dialog.tutorials.tutorialStates.ToyIntroState.react (Interpretation input)
State roboy.dialog.tutorials.tutorialStates.ToyIntroState.getNextState ()
```

Protected Functions

```
Set<String> roboy.dialog.tutorials.tutorialStates.ToyIntroState.getRequiredTransitionN
Set<String> roboy.dialog.tutorials.tutorialStates.ToyIntroState.getRequiredParameterNa
```

class

This state is meant to be used as a fallback-only state.

It only implements the react() function returning a hardcoded random answer. The react function of this state will be used if another state can't react and requires a fallback.

This state should never become active (meaning that no transition should point to it.)

ToyRandomAnswerState interface: 1) Fallback is not required (this state should be the fallback). 2) This state has no outgoing transitions. 3) No parameters are used.

Public Functions

```
roboy.dialog.tutorials.tutorialStates.ToyRandomAnswerState.ToyRandomAnswerState (String
Output roboy.dialog.tutorials.tutorialStates.ToyRandomAnswerState.act ()
Output roboy.dialog.tutorials.tutorialStates.ToyRandomAnswerState.react (Interpretation
State roboy.dialog.tutorials.tutorialStates.ToyRandomAnswerState.getNextState ()
```

class Triple

Represents a simple who(subject) does what(predicate) to whom(object) relation.

Public Functions

```
roboy.linguistics.Triple.Triple(String subject, String predicate, String object)
String roboy.linguistics.Triple.toString()
boolean roboy.linguistics.Triple.equals(Object obj)
int roboy.linguistics.Triple.hashCode()
```

Public Members

```
String roboy.linguistics.Triple.subject
String roboy.linguistics.Triple.predicate
String roboy.linguistics.Triple.object
```


class

Created by robey on 7/27/17.

Public Functions

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

Private Members

```
DatagramSocket robey.io.UdpInput.serverSocket
```

class

Created by robey on 7/27/17.

Public Functions

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

Private Members

```
DatagramSocket robey.io.UdpOutput.serverSocket
InetAddress robey.io.UdpOutput.udpEndpointAddress
int robey.io.UdpOutput.updEndpointPort
```

class Util : public Exception

Helper class.

Public Static Functions

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

enum UtteranceSentiment

Public Members

```
robey.linguistics.Linguistics.UtteranceSentiment.POSITIVE
robey.linguistics.Linguistics.UtteranceSentiment.NEUTRAL
robey.linguistics.Linguistics.UtteranceSentiment.NEGATIVE
robey.linguistics.Linguistics.UtteranceSentiment.UNCERTAIN_POS
```

```
roboy.linguistics.Linguistics.UtteranceSentiment.UNCERTAIN_NEG
```

```
roboy.linguistics.Linguistics.UtteranceSentiment.MAYBE
```

```
class Uuid
```

Public Functions

```
roboy.util.Uuid.Uuid(UuidType type, String id)
```

```
String roboy.util.Uuid.getId()
```

```
UuidType roboy.util.Uuid.getType()
```

Private Members

```
UuidType roboy.util.Uuid.type
```

```
String roboy.util.Uuid.uuid
```

```
enum UuidType
```

Public Functions

```
boolean roboy.util.UuidType.isValidUuid(String uuid)
```

```
Neo4jProperty roboy.util.UuidType.toNeo4jProperty()
```

```
Neo4jLabel roboy.util.UuidType.toNeo4jLabel()
```

Public Members

```
roboy.util.UuidType.TELEGRAM_UUID
```

```
roboy.util.UuidType.SLACK_UUID
```

```
enum UzupisIntents
```

Public Functions

```
roboy.util.UzupisIntents.UzupisIntents(String type)
```

Public Members

```
roboy.util.UzupisIntents.INTRO = ("INTRO")
```

```
roboy.util.UzupisIntents.NAME = ("NAME")
```

```
roboy.util.UzupisIntents.FRUIT = ("FRUIT")
```

```
roboy.util.UzupisIntents.COLOR = ("COLOR")
```

```
roboy.util.UzupisIntents.ANIMAL = ("ANIMAL")
```

```
roboy.util.UzupisIntents.WORD = ("WORD")
```

```
roboy.util.UzupisIntents.APPLES = ("APPLES")
roboy.util.UzupisIntents.PLANT = ("PLANT")
String roboy.util.UzupisIntents.type
```

Public Static Functions

```
static UzupisIntents roboy.util.UzupisIntents.randomIntent ()
```

Private Static Attributes

```
    Collections.unmodifiableList(Arrays.asList(values())) ]
static final int roboy.util.UzupisIntents.SIZE = VALUES.size()
static final Random roboy.util.UzupisIntents.RANDOM = new Random()
class
    A class that will issue a naturalization certificate for the Republic of Uzupiz Asks a few personalStates questions
    and can automagically generate a pdf with a certificate and print it (python script)
```

Public Functions

```
roboy.dialog.states.eventStates.UzupisState.UzupisState(String stateIdentifier, StateP
Output roboy.dialog.states.eventStates.UzupisState.act ()
Output roboy.dialog.states.eventStates.UzupisState.react (Interpretation input)
State roboy.dialog.states.eventStates.UzupisState.getNextState ()
```

Public Members

```
Interlocutor roboy.dialog.states.eventStates.UzupisState.person
```

Protected Functions

```
Set<String> roboy.dialog.states.eventStates.UzupisState.getRequiredParameterNames ()
```

Private Members

```
final Logger roboy.dialog.states.eventStates.UzupisState.logger = LogManager.getLogger()
final String roboy.dialog.states.eventStates.UzupisState.QAFILEPATH = "QAFilePath"
final String roboy.dialog.states.eventStates.UzupisState.CERTIFICATESGENERATOR = "Certifica
ArrayList<UzupisIntents> roboy.dialog.states.eventStates.UzupisState.alreadyAsked
final int roboy.dialog.states.eventStates.UzupisState.toAskCounter = UzupisIntents.values().length
UzupisIntents roboy.dialog.states.eventStates.UzupisState.currentIntent
Map<String, List<String> > roboy.dialog.states.eventStates.UzupisState.questions
```

```
Map<String, List<String> > roboy.dialog.states.eventStates.UzupisState.successAnswers
Map<String, List<String> > roboy.dialog.states.eventStates.UzupisState.failureAnswers
String roboy.dialog.states.eventStates.UzupisState.CertificatesGeneratorScript
```

```
template <V>
class Value : public roboy::context::AbstractValue<V>
    Stores a single value of type V.
    Subclassed by roboy.context.contextObjects.ActiveInterlocutor
```

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
template <V>
class
    HashMap implementation of a value history with unique Integer keys.
    Subclassed by roboy.context.contextObjects.AudioDirection, roboy.context.contextObjects.DialogIntents,
roboy.context.contextObjects.DialogTopics, roboy.context.contextObjects.ROSTest
```

Public Functions

```
roboy.context.ValueHistory< V >.ValueHistory()
V roboy.context.ValueHistory< V >.getValue() Return
    The last element added to this history.

synchronized HashMap<Integer, V> roboy.context.ValueHistory< V >.getLastNValues(int n)
    Get a copy of the last n entries added to the history.

    Less values may be returned if there are not enough values in this history. In case of no values, an empty
    array is returned.

    Return A hashmap of n last values added to the history.

Parameters
    • n: The number of instances to retrieve.

synchronized void roboy.context.ValueHistory< V >.updateValue(V value)
    Puts a value into the history in the last place.

Parameters
    • value: The value to be added.

synchronized boolean roboy.context.ValueHistory< V >.contains(V value)
    Checks if the value is contained in the history.
```

Return true if such value is in the history and false otherwise

Parameters

- `value`: The value to be checked if exists.

synchronized boolean robpy.context.ValueHistory< V >.purgeHistory()

Removes all values from the history.

Return true if the history was emptied and false otherwise

int robpy.context.ValueHistory< V >.getNumberOfValuesSinceStart()

Private Functions

synchronized int robpy.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 robpy.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 robpy.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> robpy.context.ValueHistory< V >.data

template <I, V>

class ValueInterface

This is the interface over which *Context* values can be queried.

Initialize as static field of the *Context* class. Add your *Value* implementation class and its return type as generic parameters.

Parameters

- `<I>`: An implementation of *AbstractValue*, such as the standard *Value*, ROS or Observable.
- `<V>`: The type of data stored within the *Value* instance.

Public Functions

V robpy.context.ValueInterface< I extends AbstractValue< V, V >.getValue()

Get the last element saved into the corresponding *Value* instance.

Protected Functions

```
roboy.context.ValueInterface< I extends AbstractValue< V, V >.ValueInterface(I value)
```

Protected Attributes

```
I roboy.context.ValueInterface< I extends AbstractValue< V, V >.value
```

Package Functions

```
I roboy.context.ValueInterface< I extends AbstractValue< V, V >.getContextObject()
```

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

```
new RandomList<>("roboi", "robot", "boy", "roboboy", "robot", "roboby")]
```

```
new RandomList<>("yes", "I do", "sure", "of course", " go ahead")]
```

```
new RandomList<>("no", "nope", "later", "other time", "not")]
```

```
new RandomList<>("talk", "fun", "conversation", "new", "chat")]
```

```
new RandomList <>("hello","hi","greetings", "howdy", "hey", "good evening", "what's up", "greeting  
to everyone here","hi there people", "hello world","gruse gott","wazup wazup wazup","howdy humans",  
"good evening ladies and gentlemen", "hey hey hey you there") ]
```

```
"ciao", "goodbye", "cheerio", "bye", "see you", "farewell", "bye-bye") ]
```

```
new RandomList <>("Let's go. ", "Can't wait to start. ", "Nice, let's start. ", "Yeah, let's go. " ) ]
```

```
new RandomList <>("You seem to be pretty sure about that. ", "You said yes. ", "I heard a yes from you.  
", "A clear yes. " ) ]
```

```
new RandomList <>("The chance is quite high. ", "Might be true. ", "You're not sure but probably yes. "  
) ]
```

```
new RandomList <>("Maybe you have never thought of that before. ", "Maybe, maybe not. ", "You don't  
know, no problem. ", "I don't know either. " ) ]
```

```

new RandomList <>("The chance is quite low. ", "Might be false. ", "You're not sure but probably no. ",
"Probably not. " ) ]

new RandomList <>("Not at all. ", "A clear no. ", "That was clear. No. ", "A no from you. ", "Nope. " ) ]

new RandomList <>("Oh no, I didn't get what you said. ", "I didn't understand you correctly. ", "Sorry?
What did you say? " ) ]

```

Private Functions

```

SpeechAction roboy.talk.Verbalizer.greet(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

```

new RandomList <>("talking about ", "since you mentioned ", "on the topic of ") ]

new RandomList <>("here is an interesting bit of trivia. ", "how about this? ") ]

new RandomList <>("did you know ", "did you know that ", "i read that ", "i heard that ", "have you heard
this: " ) ]

final Map<String,String> roboy.talk.Verbalizer.dayNumberMap

1,"one", 2,"two", 3,"three", 4,"four", 5,"five", 6,"six", 7,"seven", 8,"eight", 9,"nine", 10,"ten", 11,"eleven",
12,"twelve", 13,"thirteen", 14,"fourteen", 15,"fifteen", 16,"sixteen", 17,"seventeen", 18,"eighteen",
19,"nineteen" ) ]

"1","January", "2","February", "3","March", "4","April", "5","May", "6","June", "7","July", "8","Au-
gust", "9","September", "01","January", "02","February", "03","March", "04","April", "05","May",
"06","June", "07","July", "08","August", "09","September", "10","October", "11","November",
"12","December" ) ]

1,"ten", 2,"twenty", 3,"thirty", 4,"forty", 5,"fifty", 6,"sixty", 7,"seventy", 8,"eighty", 9,"ninety" ) ]

```

```
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 : public 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

This fallback state will query the generative model over ROS to create a reply for any situation.

This state is meant to be used as a fallback-only 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.)

WildTalkState interface: 1) Fallback is not required (this state should be the fallback). 2) This state has no outgoing transitions. 3) No parameters are used.

Public Functions

```
roboy.dialog.states.ordinaryStates.WildTalkState.WildTalkState(String stateIdentifier,  
Output roboy.dialog.states.ordinaryStates.WildTalkState.act()  
Output roboy.dialog.states.ordinaryStates.WildTalkState.react(Interpretation input)  
State roboy.dialog.states.ordinaryStates.WildTalkState.getNextState()  
boolean roboy.dialog.states.ordinaryStates.WildTalkState.isFallbackRequired()
```

Protected Functions

```
Set<String> roboy.dialog.states.ordinaryStates.WildTalkState.getRequiredTransitionName  
Set<String> roboy.dialog.states.ordinaryStates.WildTalkState.getRequiredParameterNames
```


Private Members

“Hey, who disconnected me from my beloved ros node? I need it! “, “Oh well, my generative model is not connected. That makes me sad. “, “Could you open a hotspot for me, I cannot connect to some services “, “I’m on holiday and don’t have internet connection right now, let’s talk about something else “)]

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 roboy.linguistics.word2vec.examples.Word2vecUptrainingExample.main(String[] args)
```

```
namespace gson
```

```
namespace api
```

```
namespace awt
```

```
namespace io
```

```
namespace net
```

```
namespace util
```

```
namespace concurrent
```

```
namespace swing
```

```
namespace query
```

```
namespace model
```

```
namespace sparql
```

```
namespace Assert
```

```
namespace models
```

```
namespace node
```

```
namespace roboy
```

```
namespace context
```

```
namespace contextObjects
```

```
namespace dialog
```

```
namespace action
```

namespace personality
namespace states
namespace botboy
namespace definitions
namespace eventStates
namespace expoStates
namespace gameStates
namespace ordinaryStates
namespace tutorials
namespace tutorialStates
namespace emotions
namespace RoboyEmotion
namespace io
namespace linguistics
namespace Linguistics
namespace phonetics
namespace sentenceanalysis
namespace word2vec
namespace examples
namespace logic
namespace memory
namespace Neo4jLabel
namespace Neo4jProperty
namespace Neo4jRelationship
namespace nodes
namespace RelationshipAvailability
namespace parser
namespace ros
namespace talk
namespace util
namespace UzupisIntents
namespace roboy_communication_cognition
namespace roboy_communication_control
file ContextIntegrationTest.java
file MemoryIntegrationTest.java
file AbstractValue.java

file **AbstractValueHistory.java**
file **Context.java**
file **ContextGUI.java**
file **ActiveInterlocutor.java**
file **ActiveInterlocutorUpdater.java**
file **AudioDirection.java**
file **AudioDirectionUpdater.java**
file **CoordinateSet.java**
file **DialogIntents.java**
file **DialogIntentsUpdater.java**
file **DialogTopics.java**
file **DialogTopicsUpdater.java**
file **FaceCoordinates.java**
file **FaceCoordinatesObserver.java**
file **IntentValue.java**
file **OtherQuestionsUpdater.java**
file **ROSTest.java**
file **ROSTestUpdater.java**
file **ExternalUpdater.java**
file **HistoryInterface.java**
file **InternalUpdater.java**
file **ObservableValue.java**
file **PeriodicUpdater.java**
file **ROSTopicUpdater.java**
file **TimestampedValueHistory.java**
file **Value.java**
file **ValueHistory.java**
file **ValueInterface.java**
file **Action.java**
file **EmotionAction.java**
file **FaceAction.java**
file **SpeechAction.java**
file **Chatbot.java**
file **Conversation.java**
file **ConversationManager.java**
file **DialogStateMachine.java**

file **DialogSystem.java**
file **Personality.java**
file **StateBasedPersonality.java**
file **Segue.java**
file **BotBoyState.java**
file **ExpoState.java**
file **State.java**
file **StateFactory.java**
file **StateParameters.java**
file **UzupisState.java**
file **DemonstrateAbilitiesState.java**
file **DemonstrateSkillsState.java**
file **ExpoIntroductionState.java**
file **PersonalInformationAskingState.java**
file **PersonalInformationAskingState.java**
file **RoboyQAState.java**
file **ChooseGameState.java**
file **GamingSnapchatState.java**
file **GamingTwentyQuestionsState.java**
file **FarewellState.java**
file **IntroductionState.java**
file **PassiveGreetingsState.java**
file **PersonalInformationFollowUpState.java**
file **QuestionAnsweringState.java**
file **WildTalkState.java**
file **StateMachineExamples.java**
file **DoYouKnowMathState.java**
file **ToyFarewellState.java**
file **ToyGreetingsState.java**
file **ToyIntroState.java**
file **ToyRandomAnswerState.java**
file **RoboyEmotion.java**
file **BingInput.java**
file **BingOutput.java**
file **CelebritySimilarityInput.java**
file **CerevoiceOutput.java**

file **CleanUp.java**
file **CommandLineInput.java**
file **CommandLineOutput.java**
file **EmotionOutput.java**
file **FreeTTSOutput.java**
file **IBMWatsonOutput.java**
file **Input.java**
file **InputDevice.java**
file **MultiInputDevice.java**
file **MultiOutputDevice.java**
file **OutputDevice.java**
file **RoboyNameDetectionInput.java**
file **TelegramInput.java**
file **TelegramOutput.java**
file **UdpInput.java**
file **UdpOutput.java**
file **Vision.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 **ProfanityAnalyzer.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 **Inference.java**
file **InferenceEngine.java**
file **StatementInterpreter.java**
file **DummyMemory.java**
file **Lexicon.java**
file **LexiconLiteral.java**
file **LexiconPredicate.java**
file **Memory.java**
file **Neo4jLabel.java**
file **Neo4jMemory.java**
file **Neo4jMemoryInterface.java**
file **Neo4jMemoryOperations.java**
file **Neo4jProperty.java**
file **Neo4jRelationship.java**
file **Interlocutor.java**
file **MemoryNodeModel.java**
file **Roboy.java**
file **Util.java**
file **Ros.java**
file **RosMainNode.java**
file **RosManager.java**
file **RosServiceClients.java**
file **RosSubscribers.java**
file **PhraseCollection.java**
file **StatementBuilder.java**
file **Verbalizer.java**
file **Agedater.java**

```
file ConfigManager.java
file FileLineReader.java
file IO.java
file JsonEntryModel.java
file Lists.java
file Maps.java
file Pair.java
file QAFileParser.java
file QAJsonParser.java
file RandomList.java
file TelegramCommunicationHandler.java
file Timeout.java
file Uuid.java
file UuidType.java
file UzupisIntents.java
file ContextTest.java
file MiniTestStateMachineCreator.java
file StateFactoryTest.java
file StateMachineEqualityTest.java
file StateMachineInitializationTest.java
file AnswerAnalyzerTest.java
file DictionaryBasedSentenceTypeDetectorTest.java
file InterpretationTest.java
file OpenNLPParserTest.java
file InferenceEngineTest.java
file QAParserTest.java
file VerbalizerTest.java
dir /home/docs/checkouts/readthedocs.org/user_builds/roboy-dialog/checkouts/latest/dialog/src
dir /home/docs/checkouts/readthedocs.org/user_builds/roboy-dialog/checkouts/latest/dialog/src
dir /home/docs/checkouts/readthedocs.org/user_builds/roboy-dialog/checkouts/latest/dialog/src
dir /home/docs/checkouts/readthedocs.org/user_builds/roboy-dialog/checkouts/latest/dialog/src
dir /home/docs/checkouts/readthedocs.org/user_builds/roboy-dialog/checkouts/latest/dialog/src
dir /home/docs/checkouts/readthedocs.org/user_builds/roboy-dialog/checkouts/latest/dialog/src
dir /home/docs/checkouts/readthedocs.org/user_builds/roboy-dialog/checkouts/latest/dialog/src
dir /home/docs/checkouts/readthedocs.org/user_builds/roboy-dialog/checkouts/latest/dialog/src
dir /home/docs/checkouts/readthedocs.org/user_builds/roboy-dialog/checkouts/latest/dialog/src
dir /home/docs/checkouts/readthedocs.org/user_builds/roboy-dialog/checkouts/latest/dialog/src
```



```
dir /home/docs/checkouts/readthedocs.org/user_builds/roboy-dialog/checkouts/latest/dialog/sr
```

```
dir /home/docs/checkouts/readthedocs.org/user_builds/roboy-dialog/checkouts/latest/dialog/sr
```

3.14 Resources

For the `config.properties` resource, please see *Configuration*.

3.14.1 JSON Resources

- personality files - contain the description of Roboy's personality:

```
{
  "initialState": "stateIdentifier",
  "states": [
    {
      "identifier": "stateIdentifier",
      "implementation": "roboy.dialog.states.StateImplementation",
      "transitions": {
        "stateTransition": "anotherStateIdentifier"
      },
      "parameters": {
        "stateParameter": "someStringParameter"
      }
    }
  ]
}
```

- question asking lists - contain the personal questions in the following form:

```
"INTENT": {
  "Q": [
    "Question phrasing 1",
    "Question phrasing 2",
    "Question phrasing 3"
  ],
  "A": {
    "SUCCESS": [
      "Possible answer on success 1",
      "Possible answer on success 2"
    ],
    "FAILURE": [
      "Possible answer on failure"
    ]
  }
}
"FUP": {
  "Q": [
    "Possible question to update the existing information"
  ],
  "A": [
    "Possible answer to the input"
  ]
}
}
```

- telegram tokens - contain tokens for registered telegram bots to allow the Dialog System to connect to the telegram API:

```
.. code:: json
{
  "TOKEN": "AAAAAAAA:AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA",
  "BOT_USERNAME": "Botname"
}
```

3.14.2 CSV Resources

- trivia - funny facts Roboy would love to tell you in the following form:

keyword;Reddit;The sentence containing the particular fact with regard to the keyword

Warning: There is no positive or negative evidence that the trivia facts work when omitting “Reddit” in the middle!

3.14.3 BIN Resources

- BIN files contain the models for the Roboy Semantic Parser

3.14.4 XML Resources

- contains the configuration for the Roboy Dialog Logger where you can set the logger scope and the means of output

3.15 Libraries and External Software

Contains a list of the libraries and external software used by this system.

3.15.1 Natural Language Understanding (nlu/roboy_parser)

Implementation

The NLU submodule is used to translate text inputs into formal semantic representations. This allows for capturing the semantic intent behind a statement or question, and using knowledge bases to translate formal question representations into answers.

The roboy_parser NLU module is based on *SEMPRE* <<http://nlp.stanford.edu/software/semprer/>>. It is currently being modified to fulfill Roboy Dialog system needs. It's architecture looks like this:

Functionalities

Roboy NLU currently has the following functionalities:

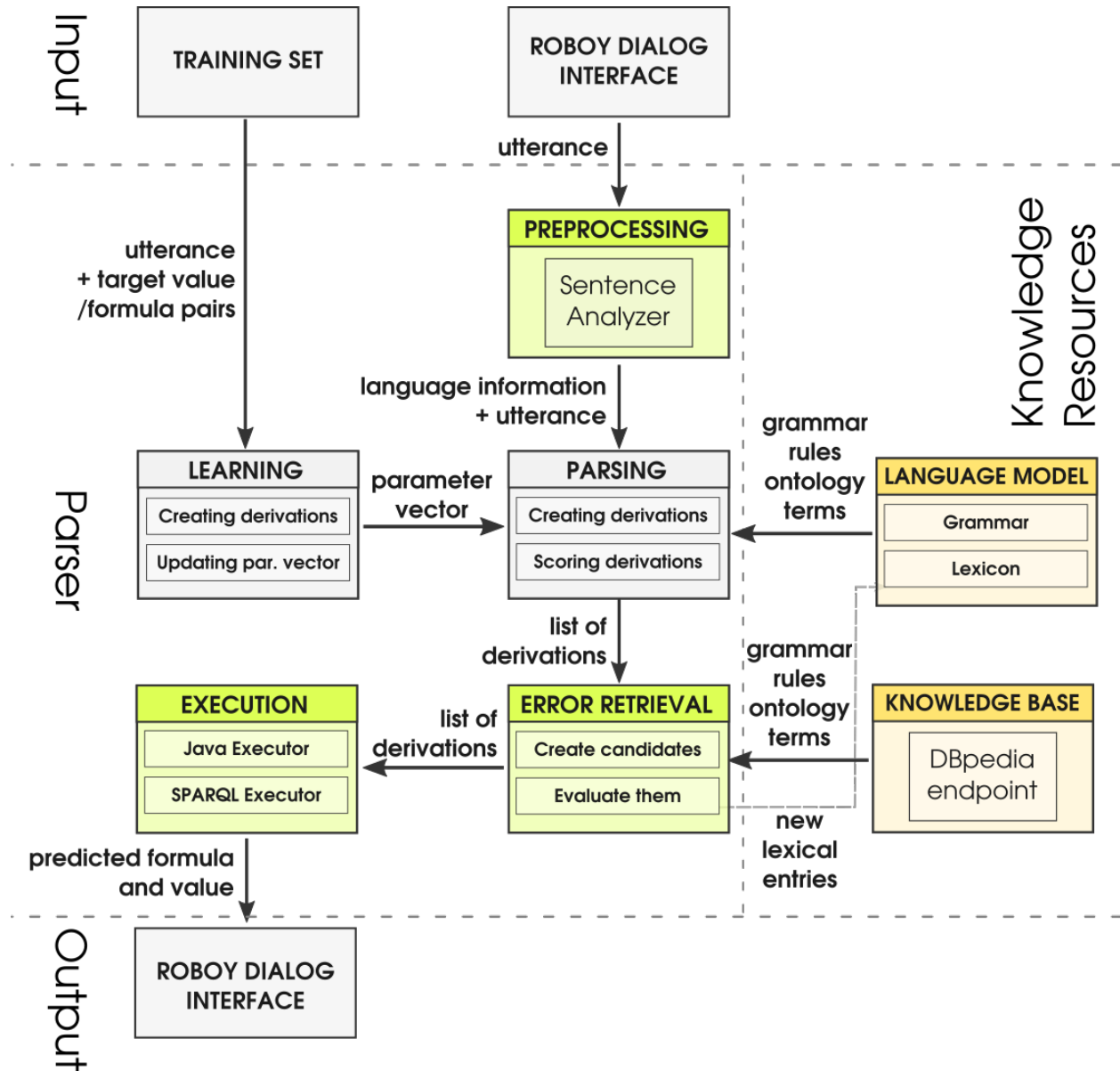


Table 3: Semantic Parser algorithms used

Functionality	Software used	Summary
Tokenization	OpenNLP	Tokenized utterance
POS Tagging	OpenNLP	Tagging tokens as part of speech
NER Tagging	OpenNLP	Tool used to tag named entities like PERSON, NUMBER, ORGANIZATION
Triplet extraction	OpenIE	Tool used to extract triples from sentences in form (Subject, Predicate, Object)
Semantic parsing	SEMPRE	Logical representation of an utterance
Semantic question answering	SEMPRE	Answer for resulting parser result
Follow-up question specification	SEMPRE	Follow-up questions for underspecified term

3.15.2 Roboy Memory

Implementation

Roboy’s Dialog System interactions with the Memory module ([learn more](#)) is done via direct function calls. The messages are sent using the methods in `roboy.memory.Neo4jMemoryOperations`, which implements the four query types based on the specified Memory services.

Note: Alternatively, one can use methods within `roboy.ros.RosMainNode` to send queries via ROS, however this is now deprecated and will require some tinkering around with code to get working. Save yourself the headache, don’t go back to this form of communication unless absolutely necessary.

Method name	Description
Create	Creates a node in Memory database
Update	Adds or changes information of an existing node
Get	Retrieves either one node or an array of IDs
Delete	Removes information from or deletes a node
Cypher	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 `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, `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.

3.15.3 Other

TelegramBots Library

To receive and send messages the library in the following github link has been used: <https://github.com/rubenlagus/TelegramBots>

Akinator Library

To receive questions and send answers during the Akinator game the following API has been used: <https://github.com/markozajc/Akiwrapper>

3.16 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.

3.16.1 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

3.16.2 Examples

- [HTML Sanity Checker](#)
- [DocChess](#) (german)
- [Gradle](#) (german)
- [MaMa CRM](#) (german)
- [Financial Data Migration](#) (german)

3.16.3 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*!

3.16.4 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), 157

E

edu::cmu::sphinx::api (C++ type), 157

J

java::awt (C++ type), 157

java::io (C++ type), 157

java::net (C++ type), 157

java::util (C++ type), 157

java::util::concurrent (C++ type), 157

javax::swing (C++ type), 157

O

org::apache::jena::query (C++ type), 157

org::apache::jena::rdf::model (C++ type), 157

org::apache::jena::sparql (C++ type), 157

org::junit::Assert (C++ type), 157

org::robby::memory::models (C++ type), 157

org::ros::node (C++ type), 157

R

robby (C++ type), 157

robby::context (C++ type), 157

robby::context::AbstractValue (C++ class), 43

robby::context::AbstractValueHistory (C++ class), 44

robby::context::Context (C++ class), 50

robby::context::ContextGUI (C++ class), 52

robby::context::ContextIntegrationTest (C++ class), 52

robby::context::contextObjects (C++ type), 157

robby::context::contextObjects::ActiveInterlocutor (C++ class), 44

robby::context::contextObjects::ActiveInterlocutorUpdater (C++ class), 44

robby::context::contextObjects::AudioDirection (C++ class), 46

robby::context::contextObjects::AudioDirectionUpdater (C++ class), 46

robby::context::contextObjects::CoordinateSet (C++ class), 56

robby::context::contextObjects::DialogIntents (C++ class), 59

robby::context::contextObjects::DialogIntentsUpdater (C++ class), 59

robby::context::contextObjects::DialogTopics (C++ class), 63

robby::context::contextObjects::DialogTopicsUpdater (C++ class), 63

robby::context::contextObjects::FaceCoordinates (C++ class), 71

robby::context::contextObjects::FaceCoordinatesObserver (C++ class), 71

robby::context::contextObjects::IntentValue (C++ class), 80

robby::context::contextObjects::OtherQuestionsUpdater (C++ class), 103

robby::context::contextObjects::ROSTest (C++ class), 126

robby::context::contextObjects::ROSTestUpdater (C++ class), 126

robby::context::ContextTest (C++ class), 53

robby::context::ContextTest::DirVec (C++ class), 64

robby::context::ExternalUpdater (C++ class), 70

robby::context::HistoryInterface (C++ class), 75

robby::context::InternalUpdater (C++ class), 82

robby::context::ObservableValue (C++ class), 101

robby::context::Value (C++ class), 152

robby::context::ValueInterface (C++ class), 153

robby::dialog (C++ type), 157

robby::dialog::action (C++ type), 157

robby::dialog::action::Action (C++ class), 44

robby::dialog::Chatbot (C++ class), 47

robby::dialog::Conversation (C++ class), 53

robby::dialog::ConversationManager (C++ class), 54

robby::dialog::DialogStateMachine (C++ class), 59

robby::dialog::DialogSystem (C++ class), 63

robby::dialog::MiniTestStateMachineCreator (C++ class), 94

robpy::dialog::personality (C++ type), 157
 robpy::dialog::personality::Personality (C++ class), 111
 robpy::dialog::Segue (C++ class), 127
 robpy::dialog::Segue::SegueType (C++ type), 128
 robpy::dialog::StateFactoryTest (C++ class), 138
 robpy::dialog::StateMachineEqualityTest (C++ class), 138
 robpy::dialog::StateMachineInitializationTest (C++ class), 139
 robpy::dialog::states (C++ type), 158
 robpy::dialog::states::botboy (C++ type), 158
 robpy::dialog::states::definitions (C++ type), 158
 robpy::dialog::states::definitions::State (C++ class), 132
 robpy::dialog::states::definitions::State::Output (C++ class), 103
 robpy::dialog::states::definitions::State::Output::OutputType (C++ type), 105
 robpy::dialog::states::definitions::StateFactory (C++ class), 138
 robpy::dialog::states::definitions::StateParameters (C++ class), 140
 robpy::dialog::states::eventStates (C++ type), 158
 robpy::dialog::states::expoStates (C++ type), 158
 robpy::dialog::states::expoStates::RoboyAbility (C++ type), 118
 robpy::dialog::states::expoStates::RoboySkillIntent (C++ type), 121
 robpy::dialog::states::gameStates (C++ type), 158
 robpy::dialog::states::ordinaryStates (C++ type), 158
 robpy::dialog::tutorials (C++ type), 158
 robpy::dialog::tutorials::StateMachineExamples (C++ class), 139
 robpy::dialog::tutorials::tutorialStates (C++ type), 158
 robpy::emotions (C++ type), 158
 robpy::emotions::RoboyEmotion (C++ type), 119, 158
 robpy::io (C++ type), 158
 robpy::io::CleanUp (C++ class), 48
 robpy::io::Input (C++ class), 79
 robpy::io::InputDevice (C++ class), 79
 robpy::io::OutputDevice (C++ class), 105
 robpy::io::Vision (C++ class), 155
 robpy::io::Vision::VisionCallback (C++ class), 156
 robpy::linguistics (C++ type), 158
 robpy::linguistics::Concept (C++ class), 49
 robpy::linguistics::DetectedEntity (C++ class), 58
 robpy::linguistics::Entity (C++ class), 67
 robpy::linguistics::Linguistics (C++ class), 91
 robpy::linguistics::Linguistics (C++ type), 158
 robpy::linguistics::Linguistics::ParsingOutcome (C++ type), 106
 robpy::linguistics::Linguistics::SemanticRole (C++ type), 129
 robpy::linguistics::Linguistics::SentenceType (C++ type), 130
 robpy::linguistics::Linguistics::UtteranceSentiment (C++ type), 149
 robpy::linguistics::phonetics (C++ type), 158
 robpy::linguistics::phonetics::PhoneticEncoder (C++ class), 111
 robpy::linguistics::phonetics::Phonetics (C++ class), 112
 robpy::linguistics::sentenceanalysis (C++ type), 158
 robpy::linguistics::sentenceanalysis::Analyzer (C++ class), 45
 robpy::linguistics::sentenceanalysis::AnswerAnalyzerTest (C++ class), 45
 robpy::linguistics::sentenceanalysis::DictionaryBasedSentenceTypeDetector (C++ class), 63
 robpy::linguistics::sentenceanalysis::Interpretation (C++ class), 82
 robpy::linguistics::sentenceanalysis::InterpretationTest (C++ class), 85
 robpy::linguistics::sentenceanalysis::OpenNLPParserTest (C++ class), 102
 robpy::linguistics::Term (C++ class), 144
 robpy::linguistics::Triple (C++ class), 148
 robpy::linguistics::word2vec (C++ type), 158
 robpy::linguistics::word2vec::examples (C++ type), 158
 robpy::linguistics::word2vec::examples::ToyDataGetter (C++ class), 146
 robpy::linguistics::word2vec::examples::Word2vecTrainingExample (C++ class), 157
 robpy::linguistics::word2vec::examples::Word2vecUptrainingExample (C++ class), 157
 robpy::logic (C++ type), 158
 robpy::logic::InferenceEngine (C++ class), 77
 robpy::logic::InferenceEngineTest (C++ class), 79
 robpy::logic::StatementInterpreter (C++ class), 140
 robpy::memory (C++ type), 158
 robpy::memory::Lexicon (C++ class), 89
 robpy::memory::LexiconLiteral (C++ class), 90
 robpy::memory::LexiconPredicate (C++ class), 91
 robpy::memory::Memory (C++ class), 92
 robpy::memory::MemoryIntegrationTest (C++ class), 92
 robpy::memory::Neo4jLabel (C++ type), 95, 158
 robpy::memory::Neo4jMemoryInterface (C++ class), 97
 robpy::memory::Neo4jMemoryOperations (C++ class), 98
 robpy::memory::Neo4jProperty (C++ type), 99, 158
 robpy::memory::Neo4jRelationship (C++ type), 100, 158
 robpy::memory::nodes (C++ type), 158
 robpy::memory::nodes::Interlocutor::RelationshipAvailability (C++ type), 117, 158
 robpy::memory::nodes::MemoryNodeModel (C++ class), 93
 robpy::memory::Util (C++ class), 149
 robpy::parser (C++ type), 158
 robpy::parser::QAParserTest (C++ class), 115
 robpy::ros (C++ type), 158

- roboy::ros::Ros (C++ class), [122](#)
- roboy::ros::RosMainNode (C++ class), [123](#)
- roboy::ros::RosManager (C++ class), [124](#)
- roboy::ros::RosServiceClients (C++ type), [125](#)
- roboy::ros::RosSubscribers (C++ type), [125](#)
- roboy::talk (C++ type), [158](#)
- roboy::talk::PhraseCollection (C++ class), [112](#)
- roboy::talk::StatementBuilder (C++ class), [139](#)
- roboy::talk::Verbalizer (C++ class), [154](#)
- roboy::talk::VerbalizerTest (C++ class), [155](#)
- roboy::util (C++ type), [158](#)
- roboy::util::Agedater (C++ class), [44](#)
- roboy::util::ConfigManager (C++ class), [49](#)
- roboy::util::FileLineReader (C++ class), [72](#)
- roboy::util::IO (C++ class), [88](#)
- roboy::util::JsonEntryModel (C++ class), [88](#)
- roboy::util::JsonModel (C++ class), [89](#)
- roboy::util::Lists (C++ class), [91](#)
- roboy::util::Maps (C++ class), [92](#)
- roboy::util::Pair (C++ class), [105](#)
- roboy::util::QAFileParser (C++ class), [113](#)
- roboy::util::QAJsonParser (C++ class), [114](#)
- roboy::util::RandomList (C++ class), [117](#)
- roboy::util::Timeout (C++ class), [144](#)
- roboy::util::Timeout::TimeoutObserver (C++ class), [145](#)
- roboy::util::Uuid (C++ class), [150](#)
- roboy::util::UuidType (C++ type), [150](#)
- roboy::util::UzupisIntents (C++ type), [150](#), [158](#)
- roboy_communication_cognition (C++ type), [158](#)
- roboy_communication_control (C++ type), [158](#)