document

This package contains GUI elements related to the designer.

actions

This package contains action listeners and utilities (e.g., dialogs) related to the designer.
Each document listener extends the abstract class “AbstractDesignerAction” which provides a method that gets the current active tab in the designer window.

terminal

This package contains GUI and logic related to our custom ROS terminal. The ProgramRegistrator class has to register each new command in order to make it supported by the terminal.

commands

This package contains the different commands that are supported by the designer. All commands extend the abstract class Command.
If the commands has to perform some communication with ROS, than it should extend RosCommand, which provides a RosPipe object used for communication with ROS.

ServiceCaller

Please pay attention to this class, which enables to call a ROS service and send/receive data. This class contains 2 useful methods:

- callService - launches a service according to the given service name and service arguments
- execute - used in the Terminal. This method calls a service and prints the output directly to the terminal.

TopicListener

This class represents a ROS topic listener. Right now, it is used only in the Terminal (prints data to the console). If one may want to use it in a general context (i.e., like ServiceCaller), he should create a default constructor (in order to avoid using the Terminal) and use the code of “execute” with another LineProcessor.

communication

This package contains the core components that perform actual communication with ROS.

RosPipe

This class represents a 2-way communication with ROS. The pipe expects to receive the following data in its constructor:

- Thread - used for interrupting the communication in middle of execution and hence to avoid blocking. In terminal context, a new thread should be created. In other context, current thread should be referenced.
- RosTargets - enum that provides the pipe an information about the desired ROS target (currently: service, topic and rossrv). The pipe transforms the enum to a valid ROS string by the private method: “convertRosEnumToString”.
- LineProcessor - output handler from the component which called the pipe.
- String... - command arguments.

Usage

After creating a pipe, the user should call the public method “sendAndReceive”. The method contains 2 phases which are interruptable from outer scope.
The method first creates a new process and sends a ROS command to ROS engine.
Afterwards, the pipe gets the process’ output and use the LineProcessor to manipulate the process’ data.

RosExecuter

This class communicates with the Executer. It is able to start/stop the execution of a behavior tree. In addition, it maintains threads that listen to the Executer current state and able to provide different services. It is able to shutdown nicely all running threads by calling “shutDownAll”.

RosStackListener

It is able listening to the trees that are being executed and color the execution path in the GUI. While listening to the stack stream of the Executer, it waits for receiving a complete message (stack frame) and color the GUI tree components.

RosStopListener

not implemented

Utils

Provides a random String generator (used for tree IDs) and for searching string matches according to regex.

lineprocessors

Provides data processors (strategy pattern) for the RosPipe object. Each has to implement the basic interface “LineProcessor”. The RosPipe contains 3 hooks that calls the LineProcessor methods.

- onStart - is called before processing data.
- onNewLine - is called when a new data line is available.
- onEnd - is called after processing data is complete.

Some useful processros:

DummyLineProcessor

Does nothing, used for providing an empty argument.

ConsoleWriterLineProcessor

Prints each output line to the Terminal

ArrayLineProcessor

Contains a group of other line processor and executes each one according the event.

BatchLineProcessor

Stores the obtained output data in an ArrayList structure, each line is stored in a separate entry.