Contents

1 Introduction 1
  1.1 What is SpyGlass? 1
  1.2 Installation 1
    1.2.1 Prerequisites 1
    1.2.2 Build instructions 2
    1.2.3 Building Spyglass as a standalone application 2
    1.2.4 Building iShell with Spyglass included as a plugin 3

2 Using SpyGlass 3
  2.1 The application interface 3
  2.2 Buttons and their behavior 5

3 SpyGlass packets 7
  3.1 Packet types 7
  3.2 Neighborhood-Packet 7
  3.3 Coordinates-List-Packet-2 8
  3.4 Coordinates-List-Packet-3 8
  3.5 Trajectory-Packet-2 8
  3.6 Trajectory-Packet-3 9
  3.7 Data-Packet 10

4 StringFormatter 11

5 SpyGlass Plugins 12
  5.1 General information about plugins 12
  5.2 SpyGlass preferences dialog 12
    5.2.1 General preferences 12
    5.2.2 Basic preferences for all plugins 13
    5.2.3 Creation of a plugin instance 14
  5.3 Change the configuration of an existing instance 14
  5.4 Plugin overview 14

6 NodePainter 16
  6.1 SimpleNodePainter 16

7 NodePositioner 18
  7.1 PositionPacketNodePositioner 18
  7.2 SpringEmbedderPositioner 19

8 RelationPainter 21
  8.1 LinePainter 21
  8.2 VectorSequencePainter 22

9 BackgroundPainter 23
  9.1 GridPainter 23
Contents

9.2  ImagePainter .................................................. 25
9.3  MapPainter ..................................................... 25
9.4  NodeSensorRange .............................................. 27
9.5  ObjectPainter .................................................. 29

10  GlobalInformation ............................................. 31
    10.1  SimpleGlobalInformation .............................. 31
1 Introduction

1.1 What is SpyGlass?

SpyGlass is an application primarily designed to visualize sensor networks on a screen. It displays a set of sensor nodes at their location on a map, as well as arbitrary data sent by the nodes, e.g. connectivity status between nodes or aggregated data collected by the nodes' sensors.

The SpyGlass application itself is a framework, designed to be extendable by plugins that visualize arbitrary things. In fact, all visualization and data aggregation that is currently done, is done by a set of bundled plugins, each fulfilling a certain functionality. As said before, the primary focus of SpyGlass is the visualization of a set of sensor nodes, forming a network. However, SpyGlass can also be used in other, nearly arbitrary scenarios, where packet based communication is used. This is because visualization is done based on network packets, which represent statistical, structural or any other arbitrary data, sent by the nodes to the SpyGlass application. Those packets are then dispatched to the various plugins, which in turn, interpret the packet, draw things onto the screen when appropriate and aggregate data contained in the packet. The whole range of visualization opportunities, delivered by the currently bundled plugins, will be explained in this document in the various chapters describing the bundled plug ins.

SpyGlass bases on the requests given by the Institute of Telematics (ITM) of the University of Lübeck and was developed by Daniel Bimschas, Sebastian Ebers, Dariush Forouher and Oliver Kleine in the context of a case study for professional product development.

1.2 Installation

1.2.1 Prerequisites

The SpyGlass application is available in two different versions. One is a plugin for the iShell\footnote{see http://www.coalesenses.com/index.php?page=ishell} software, the other is a standalone version. For both versions, a Java Runtime Environment (version 6.0 or newer) is necessary. For the iShell version one needs a platform that is supported by iShell, which is currently Windows and Linux.

Tested platforms are:

- Linux x86
1 Introduction

- Linux AMD64
- Windows (32bit)
- Mac OS X with 64bit JRE (experimentally)

1.2.2 Build instructions


Spyglass can be used in two ways:

- Either as a standalone application or
- as a plugin inside the iShell tool.

Since Spyglass uses the SWT framework, one has to decide at compile-time which platform one wants to use.

Building Spyglass is very easy. One does not need any external libraries besides a working Java SDK and "ant". The resulting jar files contain all needed libraries (via the One-JAR\textsuperscript{TM} boot-mechanism) and are directly executable.

1.2.3 Building Spyglass as a standalone application

Just run one of

    $ ant standalone-linux
    $ ant standalone-linux64
    $ ant standalone-win32
    $ ant standalone-osx

depending on the platform you want to run Spyglass on.

These commands will build the file spyglass-XXX-standalone.jar, where XXX stands for the chosen architecture. This jar file includes all necessary dependencies and can be run directly via

    $ java -jar spyglass-XXX-standalone.jar
1.2.4 Building iShell with Spyglass included as a plugin

To build iShell with SpyGlass plugin one needs to run one of the commands

$ ant ishell-linux
$ ant ishell-linux64
$ ant ishell-win32
$ ant ishell-osx

depending on the platform one wants to use.

These commands will build the file ishell-XXX-spyglass.jar, where XXX stands for the chosen architecture. This jar files includes a version of iShell, the compiled Spyglass source and all libraries both applications depend on.

Similarly, it can be run directly via

$ java -jar ishell-XXX-spyglass.jar

2 Using SpyGlass

2.1 The application interface

As mentioned before, there are two different ways to start SpyGlass. First one can start the standalone application, secondly it is a plugin for iShell. For the latter one must start iShell first and activate the SpyGlass plugin afterwards. All screenshots included in this manual are taken from SpyGlass as an iShell plugin.

On the first start of SpyGlass with an empty configuration one can see the following things (see figure 1):

- SpyGlass plugin
  As iShell contains of many plugins one must explicitly click on the tab for the SpyGlass plugin to bring SpyGlass to the front.

- Button Bar
  all functions necessary to control the behaviour of SpyGlass can be found here (check subsection 2.2 for a more detailed description).

- Drawing Area
  All visualization of the sensor network will take place here. The drawing area shows a section of the whole sensor network. This section can be
2 Using SpyGlass

Figure 1: Empty screen of the SpyGlass plugin for iShell

enlarged by zooming out or downsized by zooming in. To change the section to be shown on the drawing area without changing the zoom level, one can either use the scrollbars on the bottom and on the right side of the drawing area or hold the left mousebutton when the pointer is located on the drawing area and move the mouse to the desired direction.

- **Ruler**
  The rulers’ labeling represents the current section of the drawing area with respect to the real coordinates. Thus the labels of the ruler change automatically, whenever the drawing area is either moved or if the zoom level changes.

- **iShell Console**
  Logging console of the iShell application. iShell plugins print out log messages to the console during runtime.

**Differences in the standalone version** In the standalone version, all functionalities accessible by buttons in the button bar are also accessible via the windows’ menubar. Also, there is no iShell console in the standalone version.
2.2 Buttons and their behavior

In the following all buttons’ functionalities will be explained:

**Packet Source**  The button opens a dialog to define the current packet source. The source can either be iShell (through a TCP/IP socket connection) or a file. If the packet source is iShell, then the packets come from network. The real packet source (i.e. the packet source where iShell receives packets from) must then be configured by setting the TCP/IP preferences of iShell.

*Please note:* In the standalone version of SpyGlass it is only possible to choose a file as packet source. For more information about how to generate such a file check the reference for the output file and record buttons.

**Pause / Play**  Clicking the pause button will stop the SpyGlass application delivering the sensor nodes’ packets to the various plugins currently active. By this, the visualization will freeze for the most parts as the plugins don’t receive new packets. However, some objects on the drawing area could disappear because they time out or they could move to another position for any reason (e.g. when using a SpringEmbedderNodePositioner, see subsection 7.2). Also, plugins doing statistical calculations including timing information, such as packets per second, will therefore display falsified information during the time SpyGlass is paused.

All new packets coming in during the time SpyGlass is paused, are queued. They are delivered sequentially after the play button has been clicked. The pause button toggles to a play button when it was clicked and vice versa.

**Reset**  Clicking the reset button will cause all active SpyGlass plugins to reset. Resetting a plugin means that it forgets all aggregated data and removes all drawings from the drawing area. You could say that, after all plugins are reset, the application displays the same information as it displays directly after startup. However, the configuration of plugin instances are kept.
2 Using SpyGlass

**Output File / Record** The record button is to start the recording of incoming packets. One must specify a file, where the incoming packets will be written into. The file gets the ending “.rec” and can be used later in both the standalone application and the SpyGlass iShell plugin as a packet source. The record stops if one clicks on the record button again. During the recording time, the button is white. Otherwise it is red.

**Zoom In / Out / All** The “+” and “-” buttons with the magnifier are pretty self-explanatory. One zooms in (+) while the other zooms out (−). The third magnifier button zooms exactly to the zoom level that all current network components (nodes) are displayed on the drawing area.

Additionally to the buttons, zooming can also be done with the scrollwheel on the mouse, if the pointer is located on the drawing area.

**Preferences** The button opens the preferences dialog, where you can configure the SpyGlass application, as well as the spyglass plugin instances. All that is described in detail from chapter 5 on.
3 SpyGlass packets

As SpyGlass itself is just an application to display something, it obviously needs a source of data where it gets the information from what can or must be displayed. This information comes from the so called SpyGlass packets, which are syntactically well-defined data packets. That means, that they must be formatted according to one of the five packet types described in chapter 3.1 ff.

All SpyGlass packets consist of a header and a payload, where the header format is equal for all packet types. A header must always have a length of exactly 19 bytes and contain the information shown in table 1. Note, that the numbering of the bytes of a packet starts with 0.

<table>
<thead>
<tr>
<th>Byte no.</th>
<th>type</th>
<th>content</th>
<th>comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>0, 1</td>
<td>uint16</td>
<td>packet length (header + payload)</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>uint8</td>
<td>version</td>
<td>currently fixed to &quot;2&quot;</td>
</tr>
<tr>
<td>3</td>
<td>uint8</td>
<td>syntax type</td>
<td>0 = std</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>1 = uint8List</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>2 = uint16List</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>3 = Int16List</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>4 = UInt32List</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>5 = Int64List</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>6 = FloatList</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>7 = variable</td>
</tr>
<tr>
<td>4</td>
<td>uint8</td>
<td>semantic type</td>
<td></td>
</tr>
<tr>
<td>5, 6</td>
<td>uint16</td>
<td>sender ID</td>
<td></td>
</tr>
<tr>
<td>7, 8, 9, 10</td>
<td>uint32</td>
<td>current time in seconds</td>
<td></td>
</tr>
<tr>
<td>11, 12</td>
<td>uint16</td>
<td>milliseconds of current time</td>
<td></td>
</tr>
<tr>
<td>13, 14</td>
<td>int16</td>
<td>x-coordinate of senders position</td>
<td></td>
</tr>
<tr>
<td>15, 16</td>
<td>int16</td>
<td>y-coordinate of senders position</td>
<td></td>
</tr>
<tr>
<td>17, 18</td>
<td>int16</td>
<td>z-coordinate of senders position</td>
<td></td>
</tr>
</tbody>
</table>

Table 1: SpyGlass packet header

3.1 Packet types

3.2 Neighborhood-Packet

The payload of a Neighborhood-Packet contains a list of uint16 values which represents a list of node IDs. Its syntax type is consequently “uint16List”. The list must neither contain a node ID twice, nor the node ID of the sender itself. See an overview on the payload in table 2.
3 SpyGlass packets

### Table 2: Payload of a Neighborhood-Packet

<table>
<thead>
<tr>
<th>Byte no.</th>
<th>Content</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>19, 20</td>
<td>node ID</td>
<td>uint16</td>
</tr>
<tr>
<td>20, 21</td>
<td>node ID</td>
<td>uint16</td>
</tr>
<tr>
<td>22, 23</td>
<td>node ID</td>
<td>uint16</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

### 3.3 Coordinates-List-Packet-2

A Coordinates-List-Packet-2 contains a payload, which consists of a list of 2-dimensional coordinates. Each coordinate is represented by an int16 value. Thus a 2-dimensional position needs two values and consequently the payload must have a length of a multiple of four bytes. Obviously, its syntax type is “int16List”. A tabular overview is given in table 3.

<table>
<thead>
<tr>
<th>Byte no.</th>
<th>Content</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>19, 20</td>
<td>x-coordinate of position 1</td>
<td>int16</td>
</tr>
<tr>
<td>21, 22</td>
<td>y-coordinate of position 1</td>
<td>int16</td>
</tr>
<tr>
<td>23, 24</td>
<td>x-coordinate of position 2</td>
<td>int16</td>
</tr>
<tr>
<td>25, 26</td>
<td>y-coordinate of position 2</td>
<td>int16</td>
</tr>
<tr>
<td>27, 28</td>
<td>x-coordinate of position 3</td>
<td>int16</td>
</tr>
<tr>
<td>29, 30</td>
<td>y-coordinate of position 3</td>
<td>int16</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

### Table 3: Payload of a Coordinates-List-Packet-2-Packet

### 3.4 Coordinates-List-Packet-3

A Coordinates-List-Packet-3 is very similar to a Coordinates-List-2-Packet. The only difference is the dimensionality, because this packet contains positions with three dimensions instead of 2. As each coordinate is represented by an int16 value and a 3-dimensional position needs three values, the payload must have a length of a multiple of six bytes. The syntax type is “int16List”. See table 4 for further details.

### 3.5 Trajectory-Packet-2

The payload of Trajectory-Packet-2 represents the way of an object via several 2-dimensional positions and the time it needs to come from one position to the next. So the payload contains a list of tuples, where the first two elements
### 3.6 Trajectory-Packet-3

The payload of a Trajectory-Packet-3 is very similar to a Trajectory-Packet-2. The only difference is, that the positions are not given in 2D but in 3D. Thus each position consists of three coordinates. Between two positions the payload contains the duration, the objects needs to pass the way from one position to the next.

<table>
<thead>
<tr>
<th>Byte no.</th>
<th>content</th>
<th>type</th>
</tr>
</thead>
<tbody>
<tr>
<td>19, 20</td>
<td>x-coordinate of position 1</td>
<td>int16</td>
</tr>
<tr>
<td>21, 22</td>
<td>y-coordinate of position 1</td>
<td>int16</td>
</tr>
<tr>
<td>23, 24</td>
<td>z-coordinate of position 1</td>
<td>int16</td>
</tr>
<tr>
<td>25, 26</td>
<td>x-coordinate of position 2</td>
<td>int16</td>
</tr>
<tr>
<td>27, 28</td>
<td>y-coordinate of position 2</td>
<td>int16</td>
</tr>
<tr>
<td>29, 30</td>
<td>z-coordinate of position 2</td>
<td>int16</td>
</tr>
<tr>
<td></td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>19 + 6(n-1), 20 + 6(n-1)</td>
<td>x-coordinate of position n</td>
<td>int16</td>
</tr>
<tr>
<td>21 + 6(n-1), 22 + 6(n-1)</td>
<td>y-coordinate of position n</td>
<td>int16</td>
</tr>
</tbody>
</table>

Table 4: Payload of a Coordinates-List-Packet-3-Packet

The payload of a Trajectory-Packet-3 is very similar to a Trajectory-Packet-2.

of each tuple represents a position. The third element represents the time in seconds, that the object spends between the position given before and the next position. Although the syntax type is “int16List” and all elements are of type int16, the time element must not contain negative values.

Since the last position specifies the ending point, the last tuple is incomplete. It contains only the two elements representing the position. Table 5 shows the content of the payload.

<table>
<thead>
<tr>
<th>Byte no.</th>
<th>content</th>
<th>type</th>
</tr>
</thead>
<tbody>
<tr>
<td>19, 20</td>
<td>x-coordinate of position 1</td>
<td>int16</td>
</tr>
<tr>
<td>21, 22</td>
<td>y-coordinate of position 1</td>
<td>int16</td>
</tr>
<tr>
<td>23, 24</td>
<td>duration of moving between position 1 and 2</td>
<td>int16</td>
</tr>
<tr>
<td>25, 26</td>
<td>x-coordinate of position 2</td>
<td>int16</td>
</tr>
<tr>
<td>27, 28</td>
<td>y-coordinate of position 2</td>
<td>int16</td>
</tr>
<tr>
<td>29, 30</td>
<td>duration of moving between position 2 and 3</td>
<td>int16</td>
</tr>
<tr>
<td></td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>19 + 6(n-1), 20 + 6(n-1)</td>
<td>x-coordinate of position n</td>
<td>int16</td>
</tr>
<tr>
<td>21 + 6(n-1), 22 + 6(n-1)</td>
<td>y-coordinate of position n</td>
<td>int16</td>
</tr>
</tbody>
</table>

Table 5: Payload of a Trajectory-Packet-2-Packet

3.6 Trajectory-Packet-3
3 SpyGlass packets

The syntax type is “int16List”. The duration value must be given in seconds and must not contain a negative number even though it is an int16 field. See table 6 for more details.

<table>
<thead>
<tr>
<th>Byte no.</th>
<th>content</th>
<th>type</th>
</tr>
</thead>
<tbody>
<tr>
<td>19, 20</td>
<td>x-coordinate of position 1</td>
<td>int16</td>
</tr>
<tr>
<td>21, 22</td>
<td>y-coordinate of position 1</td>
<td>int16</td>
</tr>
<tr>
<td>23, 24</td>
<td>z-coordinate of position 1</td>
<td>int16</td>
</tr>
<tr>
<td>25, 26</td>
<td>duration of moving between position 1 and 2</td>
<td>int16</td>
</tr>
<tr>
<td>27, 28</td>
<td>x-coordinate of position 2</td>
<td>int16</td>
</tr>
<tr>
<td>29, 30</td>
<td>y-coordinate of position 2</td>
<td>int16</td>
</tr>
<tr>
<td>31, 32</td>
<td>z-coordinate of position 2</td>
<td>int16</td>
</tr>
<tr>
<td>33, 34</td>
<td>duration of moving between position 2 and 3</td>
<td>int16</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>19 + 8(n-1), 20 + 8(n-1)</td>
<td>x-coordinate of position n</td>
<td>int16</td>
</tr>
<tr>
<td>21 + 8(n-1), 22 + 8(n-1)</td>
<td>y-coordinate of position n</td>
<td>int16</td>
</tr>
<tr>
<td>23 + 8(n-1), 24 + 8(n-1)</td>
<td>z-coordinate of position n</td>
<td>int16</td>
</tr>
</tbody>
</table>

Table 6: Payload of a Trajectory-Packet-3-Packet

3.7 Data-Packet

A Data-Packets payload contains arbitrary data. Its syntax type is flexible but must be defined in the header of the packet. The possible syntax types are “uint8List”, “uint16List”, “int16List”, “uint32List”, “int64List” and “floatList”. The length of the payload depends on the syntax type.
4 StringFormatter

The StringFormatter is a utility which is used by several plugins to write specific information on the screen. A StringFormatter expression combines arbitrary text with information from SpyGlass-packets. The placeholders in these expressions define the type and the position of the value in the packet. Such a placeholder always starts with a % followed by specific letter, that represents the type of the value. All possible value types and the corresponding letter are shown in table 7. Afterwards the offset of the value in the packet must be given to complete the placeholder. If the string should contain a %, it must be masked as %%. A line break must be masked as \n.

<table>
<thead>
<tr>
<th>letter</th>
<th>type</th>
</tr>
</thead>
<tbody>
<tr>
<td>u</td>
<td>uint8 (unsigned, 1 byte)</td>
</tr>
<tr>
<td>v</td>
<td>uint16 (unsigned, 2 bytes)</td>
</tr>
<tr>
<td>U</td>
<td>uint32 (unsigned, 4 bytes)</td>
</tr>
<tr>
<td>i</td>
<td>int16 (signed, 2 bytes)</td>
</tr>
<tr>
<td>f</td>
<td>float (signed, 4 bytes)</td>
</tr>
</tbody>
</table>

Table 7: Value types for a StringFormatter expression

In a packet a signed 16 bit integer must be provided in two’s complement, while a float value must be given conform to norm IEEE 754 (IEEE Standard for Binary Floating-Point Arithmetic for microprocessor systems (ANSI/IEEE Std 754-1985)).

In the following an example is given. Assume a packet with the following content (encoded in hexadecimal):

E7.DF.A0.00.00.00.32.C8.88.03.EF.89

Then the StringFormatter expression

Temp: %U3 Grad\n Battery: %u9 V

would result into the following String:

Temp: 50 Grad
Battery: 3 V
5 SpyGlass Plugins

5.1 General information about plugins

Each plugin defines a specific functionality. To use this functionality one must create instances of the plugin. Some plugins do some background computing (e.g. PositionPacketNodePositioner), while others display something on the screen (e.g. SimpleNodePainter).

Usually there will be more than one visible instance to be displayed on the screen. Thus, one needs to define a ranking, which instance should be displayed on top of another. So some content of an instance in the background might be hidden by content of another instance. This is done in the Plugin overview of the SpyGlass preference dialog (see 5.4).

But there is also another advantage for the existence of this ranking. Some plugins may provide actions, if the user clicks on an object on the screen, e.g. the NodePainter opens or closes an extended view when the double-clicks on the node. It is always the topmost plugin instance that can handle those events, that catches the command and indeed handles the event. Since currently only the NodePainter handles events, this feature is more or less for future use.

5.2 SpyGlass preferences dialog

5.2.1 General preferences

On the page for general preferences (see 3) one must define basic values especially for the relation between the position that the nodes send and their position on the screen. The unit of length is the unit that will be displayed on the preference pages of all plugins where parameters concerning any height or width must be set. It has no semantic effect, since SpyGlass expects all positions given with the packets in coordinates and this coordinates are generally interpreted unitless.

The scaling factor defines the scaling of coordinates per dimension. If e.g. an x-coordinate of a position received from a packet is 5 and the scaling factor of the x-dimension is 20, SpyGlass takes 100 as the real x-coordinate. The scaling factors of the two dimensions may be defined independently or may also associated to each other by activating the button with the chain. Then a change of one scaling factor automatically changes the scaling factor of the other dimension to the same value.

The offset defines the origin of the coordinates. The value of the x-axis is added to all x-coordinates received from a packet. It is the same for the offset of the y-axis and y-coordinates. If e.g. a packet contains a position (10,10), the x-offset is set to 50 and the y-offset is set to 80, then SpyGlass takes (60,90) as the real
5.2 SpyGlass preferences dialog

coordinates.

The real coordinates per dimension can be seen in the ruler on the drawing area. The ruler can be set to visible or invisible on the top of this page.

The button to restore the defaults sets all values to predefined default values. That is “m” as metric unit, 1 for the scaling and 0 for the offset in both dimensions.

The metric of time values are not used anywhere yet. So they are for future use. All values with respect to time have predefined time units in all current plugins.

5.2.2 Basic preferences for all plugins

There is a group of basic preferences which must be configured for all plugin instances. This is the instances name, at least one semantic type (if the plugin computes packages), the activity status and the visibility.

The name is used to reference to the instance, e.g. in dialog windows. The semantic types define, which packets will be delivered to this instance. “-1” means, that the instance will receive all packets independently from the semantic type. One can define a comma separated list of semantic types, a range or a combination of both, e.g. “3-5,8” is the same as “3,4,5,8”.

If an instance is not able to compute a packet, because the payload is not suitable to the plugin, a warning or error window will appear at runtime.
5 SpyGlass Plugins

An active plugin instance will receive all packets of the semantic types defined above. Thus, generally spoken “active” means, it computes packets. An instance does only display anything, if “visible” is set as well. So, if an instance is active but not visible, it computes the packets in the background.

5.2.3 Creation of a plugin instance

Each plugin has a particular page to create instances. On the bottom of all of these pages, one can find the buttons shown in figure 4.

![Figure 4: Buttons on the page to create a plugin instance](image)

On this page one must configure the instance that shall be created. The configuration can be saved as the new default configuration. The values of the default configuration will be set on this page whenever it is shown again or the button “restore default” is selected. A click on the button “create instance” creates an instance of the plugin with the specified configuration.

If one tries to enter illegal values into the configuration, the changes cannot be applied and a window pops up to tell the user about the mistake.

5.3 Change the configuration of an existing instance

Each plugin instance gets its own entry in the plugin tree of the preference dialog window. Here one can either delete the instance or change its configuration. If the button “apply” is pressed, then the new configuration will be activated. If one made a mistake while changing the configuration, the old values can be restored with the button “restore values” until “apply” has been pressed.

![Figure 5: Buttons on a plugin instance preference page](image)

Like on the page for the instance creation, illegal values cannot be applied.

5.4 Plugin overview

The plugin overview page presents all plugin instances in a tabular form (see 6). Here one can do some basic operations on the instances. At first one can
change the order of the plugin. Each instance that is on top of another in this list will draw its content over all instances contents that should be displayed on the same position on the screen.

Figure 6: Plugin instance overview

If e.g. an instance SNP of the SimpleNodePainter paints a node as a rectangle and a VectorSequencePainter instance VSP draws a line that crosses this rectangle, the line is drawn “behind” the node (i.e. is invisible) if SNP is above VSP in the list. Otherwise the line would be drawn above the rectangle. Thus especially BackgroundPainters should be on the bottom of this list in most cases. To change an instances position one must simply select the instance and click on “Up” or “Down”.

The button “Reset instance” makes the instance to forget all past packets. It has the same effect like deleting the instance and creating a new one with the same values. “Delete instance” deletes the selected instance unrecoverably.

The next two buttons are to change the status of an instance. There labels change automatically on demand.
6 NodePainter

6.1 SimpleNodePainter

The SimpleNodePainter is the main plugin to visualize a sensor network, since it is responsible for the illustration of the nodes themselves. The configuration page can be seen in figure 7.

Since there are two ways to illustrate a node ("normal" and "extended"), one must define which is the default one. The normal way to display a node is just to write its ID into a rectangle. Color and line width of this rectangle can be configured as well. The extended illustration shows some additional information which must be defined in the common string formatter field and the table below. The way how to use the string formatter is described in chapter 4.

Figure 7: SimpleNodePainter preference dialog

Figure 8 shows a small network with three nodes. Node 1 is displayed in normal mode, while node 2 and 3 are in extended mode. As one can see, the common string formatter is responsible for the illustration of the time in the extended mode while the speed and the temp values are specific to a semantic type.

It is possible to switch between the display modes with a double-click on the node. If a node is in front of another and one wants to bring the node from the background to the foreground, this can be done with a right-mouse-button-click on the area of the node in the background. This includes the hidden area of the background node. In case there is more than one button in the background of another, this click will bring the node to the foreground, that was the furthest in the background.
6.1 SimpleNodePainter

The SimpleNodePainter does not need metric information, since it displays the node just at the position that the active NodePositioner provides. This can be either an instance of the PositionPacketNodePositioner or the SpringEmbedder-Positioner. Both are described in the following chapter.
7 NodePositioner

A NodePositioner collects and provides the information about the current positions of the nodes in the network. Every incoming packet will be forwarded to the active NodePositioner instance. This instance grabs the nodes ID, its current position and the time, when the node sent the packet out of the packets header.

This information is available either until it is overwritten because the node sent a new packet or until the information times out. The time-out-interval must be set in the preference page of the instance. On the preference page one can decide, whether the instance is active or not. Since there can only be one active instance of the NodePositioner plugin, the current instance will be deactivated, when another instance is activated. The option “visible” cannot be set or changed since the plugin instance does not display anything anyway.

There are two types of the NodePositioner plugin which are described in the following chapters 7.1 and 7.2.

7.1 PositionPacketNodePositioner

An instance of the PositionPacketNodePositioner plugin provides the exact absolute positions for all nodes, who sent a packet in the past and whose information did not time out yet.

Additionally to the activation status one must define the timeout interval. The interval is set in seconds where 0 means, that the timeout interval is infinite. Figure 9 shows the preference dialog for the PositionPacketNodePositioner plugin.

![Figure 9: PositionPacketNodePositioner preference dialog](image)

18
7.2 SpringEmbedderPositioner

The SpringEmbedderPositioner does not provide absolute positions of the nodes. Instead of absolute positions a SpringEmbedderPositioner orders the nodes in such a way, that nodes being in a neighborhood relationship attract each other while the others repulse. For this reason the positions change dynamically.

All attraction and repulsion forces are being computed for all nodes in the network one after another once a second. Thus, the positions of all nodes might change permanently, since the distance between the nodes and thus the forces of attraction and repulsion change in every round.

Additionally to these positions, the SpringEmbedderPositioner stores the current absolute position. So, if an instance of the PositionPacketNodePositioner plugin becomes active after a SpringEmbedder instance, the new instance will be initiated with the absolute positions of all known nodes.

If a SpringEmbedderPositioner instance is being activated while another NodePositioner is running, the new instance will be provided with all nodes and its absolute positions by the old instance. Each of these nodes gets a random position as a start position for the dynamic computing. The same is valid for new nodes during the running time of the instance.

There are several values to configure for a SpringEmbedderPositioner instance which is shown in figure 10. Like a PositionPacketNodePositioner the SpringEmbedderPositioner has time-to-live for all nodes. If a node does not send a packet for a duration which is longer than this time-to-live-interval, it will be removed.

Figure 10: SpringEmbedderPositioner preference dialog

The most important value is/are the semantic type(s) of the packets, that provide the neighborhood information. If a node sends such a neighborhood packet (see chapter 3.2), it will be attracted by all its neighbors contained in the pack-
7 NodePositioner

gets payload. This does not include the reverse, so a sender of the packet does not attract all its neighbors. A node A is only being attracted by another node B, if node A is contained in a neighborhood packet sent by node B.

The other configuration values are algorithm specific. The optimum spring length represents the distance, where the attraction force between to nodes comes into a resting state. The spring stiffness defines, how fast the positions of the nodes converge to this resting state. The value must be greater than 0 and not greater than 1.

The repulsion factor defines the dimension of repulsion between all nodes, while the efficiency of forces regulates the speed of the nodes movement. It must be greater than 0 and not greater than 1. Since these values are very network specific, they should be fitted to the current requirements on demand.

The example given in figure 11 bases on a network where nodes 1, 2 and 3 have a neighborhood relationship to the node 100 and vice versa. This is also true for 1 and 10-19, 2 and 20-29 and 3 and 30-39. Thus e.g. node 100 has three neighbors, node 2 has 11 neighbors and node 20 has one neighbor. The figure shows the final resting position of the nodes. Actually there is no resting position since the algorithm for repositioning the nodes never comes to an end. But the changes decrease as longer the algorithm runs.

Since the PositionPacketNodePositioner provides absolute positions, it provides metrical information. The SpringEmbedderPositioners positions are not metric. Since some plugins can only display expedient information if the active NodePositioner instance provides its positions in such a metric way, all those instances are deactivated automatically when a SpringEmbedderPositioner instance is get-
ting active. A warning window appears before the automatic deactivation, so the operation can be stopped. If a plugin needs metrical information or not is described in the appropriate chapter.

8 RelationPainter

8.1 LinePainter

The LinePainter generally spoken draws a line between two nodes. The LinePainter needs DataPackets containing a uint16 value at the beginning. This uint16 value defines the neighbor where the line must be drawn to. All following payload starting at byte 22 (respectively offset 21) of the packet is arbitrary data and can be displayed using StringFormatter expressions.

The time to live defines how many seconds a line will be displayed. If the time to live is set to 0, the line will never be removed.

Figure 12 shows an example configuration while figure 13 shows the result of this configuration.

As one can see, the strings generated by the StringFormatter expressions are being displayed next to the sender and above the line between the sender and its neighbor.

The LinePainter does not need metric positions. It simply draws a line between two nodes.
8 RelationPainter

8.2 VectorSequencePainter

A VectorSequencePainter instance draws a sequence of lines. These lines are completely independent from node positions but could be drawn between arbitrary points. The starting point, edges and endpoint are given as coordinates via the payload of a Coordinates-List-Packet-2 (see 3.3) or a Coordinates-List-Packet-3 (see 3.4).

This dimensionality of the semantic types, the instance is registered for, must be configured in the preference page which is shown in figure 14. All of these semantic types for an instance must have the same dimensionality. Anyway, in case of 3D, the third dimension will be ignored for the illustration.

All other values in the preference page are pretty self-explanatory.

As one can see in picture 15, the position of the lines is completely independent from the position of the sending node. In this example the data for the line sequence was sent by node 150.

The VectorSequencePainter supports metric, since the starting and ending points of the lines are given in coordinates with respect to the actual positions in the real world. An instance of the VectorSequencePainter can not be active while a SpringEmbedderPositioner instance is active as well. Thus it will be deactivated automatically when a SpringEmbedderPositioner instance is getting active.
9 BackgroundPainter

9.1 GridPainter

An instance of the GridPainter plugin draws a grid with the configured dimensions onto the screen. One can configure the number of rows and columns independently or associate one to the other with a click on the button with the chain. If they are associated, then both will have the same value automatically.
9 BackgroundPainter

It is the same with the width and the height of a grid element. The preference page for an instance of the GridPainter plugin is shown in figure 16. Since a GridPainter instance does not compute any packets, no semantic type can be set. To make the grid visible, both active and visible must be set. If one of those is not set, the is not being displayed.

Figure 16: GridPainter preference dialog

The grid corresponding to the configuration above can be seen in figure 17.

Figure 17: GridPainter

A GridPainter instance does not support metric. It simply draws a grid at the specified position and the with the specified size on the drawing area. Thus an
instance can be active independently from the type of the active NodePositioner. If a SpringEmbedderPositioner instance is active, one must ignore the metric units on the preference page.

9.2 ImagePainter

The ImagePainter is to draw an image on the background. This might be an arbitrary image from the local file system. Supported images types are “jpg”, “gif” and “png”. See the preference page in figure 18.

The position is set by setting the lower left point in absolute coordinates. The image can be resized either independently per dimension or with keeping the proportions automatically when changing the value for one dimension.

![ImagePainter preference dialog](image18.png)

Figure 18: ImagePainter preference dialog

The ImagePainter does not support metric. It just draws the specified image at the specified position with the specified size. If a SpringEmbedderPositioner instance is active, one must ignore the metric units on the preference page.

9.3 MapPainter

A MapPainter instance draws a map on the specified sector of the drawing area. This sector must be defined by setting its lower left point, and its horizontal and vertical comprehensiveness on the preference page (see 19). The granularity defines the resolution of the map, i.e. the number of independently computed values (rectangles on the map). The two parameters per dimension for comprehensiveness and resolution can be associated to each other, i.e. set to the same value automatically, by using the appropriate button with the chain.

The other parameters define the range of data, that is contained in the packets
payload. Concerning this payload, two things must be clarified at the beginning. The MapPainter always uses only the first element of the payload. If the payload contains for example an int16List, only the first two bytes of the payload are read and used as an int16 value. The rest of the payload will be ignored. The other thing is, that generally all packets from nodes, whose position is outside of the maps area will be ignored as well.

The MapPainter associates a color with all possible values. If a value is less than the minimum value from the preference page it gets the same color as the minimum value. It is the same with values greater than the maximum value from the preference page. They will be associated with the same color as the maximum value. All values in between will be associated with a mixed color between the defined colors for minimum and maximum value.

Assume the maximum value is associated with black and the minimum value is associated with white. Then a value close to but lower than the maximum value will be associated with a dark grey, while a value close to but higher than the minimum value will be associated with a bright grey.

For each rectangle of the map, the appropriate value is computed by identifying the k nearest neighbors, i.e. the k nearest positions, where a measured value is known and calculate the mean of these k values.

Each map has frame points. Frame points are invisible points on the maps.
border, that behave like nodes with a fixed position and measured data. The positions of the frame points are uniformly distributed on the frame, thus per dimension the distance between neighbored frame points is equal. Each frame point provides the default value. The update frequency defines how often the map is being recomputed per second. As the algorithm to recalculate all the values is computational expensive, the actual update frequency could be slower than the configured.

Figure 20 shows an example for a MapPainter instance together with an active and visible SimpleNodePainter instance. So one can see, where the nodes are located in the maps sector.

![Figure 20: MapPainter](image)

The MapPainter supports metric, since the whole map bases on the distance of the nodes in the real world. Thus an instance can not be active while a SpringEmbedderPositioner instance is active, too.

### 9.4 NodeSensorRange

An instance of the NodeSensorRange plugin illustrates the area next to the node, that can be covered by a sensor. One can define many values for a NodeSensorRange instance. The preference page can be seen in figure 21.

In addition to the usual graphical parameters (line width, color, etc.) one must define the form. Possible options are “Circle”, “Rectangle” and “Cone”. The reference point for the illustration of the sensor range is the position of the node, i.e. the radius of a circle starts here and the center of a rectangle is here.
as well. The edge with the acute angle of a the cone is also located at this position.

Figure 21: NodeSensorRange preference dialog

One must define a default range, which will be displayed for all nodes that do not have a specific configuration in the table below. The specific configuration for the type of the range can be opened by clicking on the options button next to the selection field for the type.

The only options for a circle range is the radius. It is given with respect to the generally defined metric unit. A rectangle needs three parameters. The width and the height that must be defined with respect to the generally defined metric unit as well. The orientation value defines the angle in degrees how far the rectangle is turned to the right. A cone has three specific parameters, too. The first parameter is the length, e.g. the radius of the imaginary circle around the node, where the cone is a part of. Like for rectangles, the orientation defines the angle in degrees how far the cone is turned to the right. If the orientation angle is 0, the right edge of the cone is parallel to the x-axis. The view angle defines how wide the range is in degrees, i.e. a half circle would be 180 degrees.

See the illustration of an example instance of the NodeSensorRange plugin in figure 22.

The NodeSensorRange plugin does not support metric. Thus it can be active together with an active SpringEmbedderPositioner instance. In case of an active SpringEmbedder, the range information is more informal and not absolute.
9.5 ObjectPainter

An ObjectPainter instance draws a moving object on the drawing area. The object to be moved on the screen must be defined in the field “image file” on the preference page (see 23).

The drawing options must be set below. At first one can resize the image either with keeping the proportions or by setting width and height independently. The trajectory is the way the object takes on the screen. This way can be drawn on the screen with the specified color.

The trajectory is given by the payload of a Trajectory-Packet. This can either contain 2D or 3D coordinates describing the way of the object. The packets are explained in the chapters 3.5 and 3.6. All semantic types the plugin instance is registered for, must be of the same type. To mix both trajectory types in one network, one must define at least two ObjectPainter instances. Anyway in case of 3D, the third dimension is ignored for illustration.

The last parameter is the update interval. As smaller this interval is, the more often the plugins illustration is updated, e.g. the more smooth is the movement of the object.

The example in figure 24 shows one trajectory but many little men moving on the line. Each of this little men exists because of one packet, e.g. each packet generates a moving object on the screen. In this example, all objects take the same way but this is not necessary. An object takes the way, that is defined by...
9 BackgroundPainter

![ObjectPainter preference dialog](image1)

**Figure 23:** ObjectPainter preference dialog

![ObjectPainter](image2)

**Figure 24:** ObjectPainter

its appropriate tracetry packet.

The plugin supports metric, so an instance can not be active while an instance of the SpringEmbedderPositioner is active as well.
10 GlobalInformation

10.1 SimpleGlobalInformation

An instance of the SimpleGlobalInformation plugin writes some predefined statistical information on the right side of the screen.

Each visible instance of a SimpleGlobalInformation plugin displays the number of packets that have been sent in the network since the instance is active. The next default parameters are the numbers of packets, that have been received and computed during several periods. PPS means packets per second. The three values afterwards represent the mean number of packets per second during the last second, the last 30 seconds and the last 60 seconds. This values will always be shown and can not be configured.

The display of some standard information can be directly activated by selecting the appropriate box at the bottom of the preference page (see 25). These standard informations are the number of nodes in the network and the average node degree, i.e. the mean number of neighbors. For this reason there must be one or more semantic type(s) defined, that contain the neighborhood information (see 3.2).

![SimpleGlobalInformation preference dialog](image)

Figure 25: SimpleGlobalInformation preference dialog

The table in the middle of the preference page is for the configuration of some specific statistical information. Each row in this table belongs to one statistical
value. In the example configuration on figure 25 the packets of semantic type 20 contain a temperature value of type uint8 at the beginning of the packets payload (offset 19). This data is used to compute some statistics on it. Possible operations are “sum”, “minimum”, “maximum”, “average” and “median”.

Figure 26 shows an active instance of the SimpleGlobalInformation plugin together with an active MapPainter. As the MapPainter draws a map from the temperatures, the SimpleGlobalInformation instance computes some statistics on the same data.