DECA-PDoA-RTLS GUI SOURCE CODE GUIDE

Version 1.1

This document is subject to change without notice.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

1 INTRODUCTION

This document describes the Deca-PDoA-RTLS application and its source code. The application connects to the PDoA node (running the PDoA node application), and consumes the range and phase difference (PD) and x,y position reports coming from the PDoA node and plots tag's location estimate as seen from the node's point of view. Figure 1 shows the demo system.

The reader is encouraged to read the accompanying PDoA Evaluation kit documents, in particular the PDoA-Node Source Code Guide [1], PDoA-Tag Source Code Guide [2].

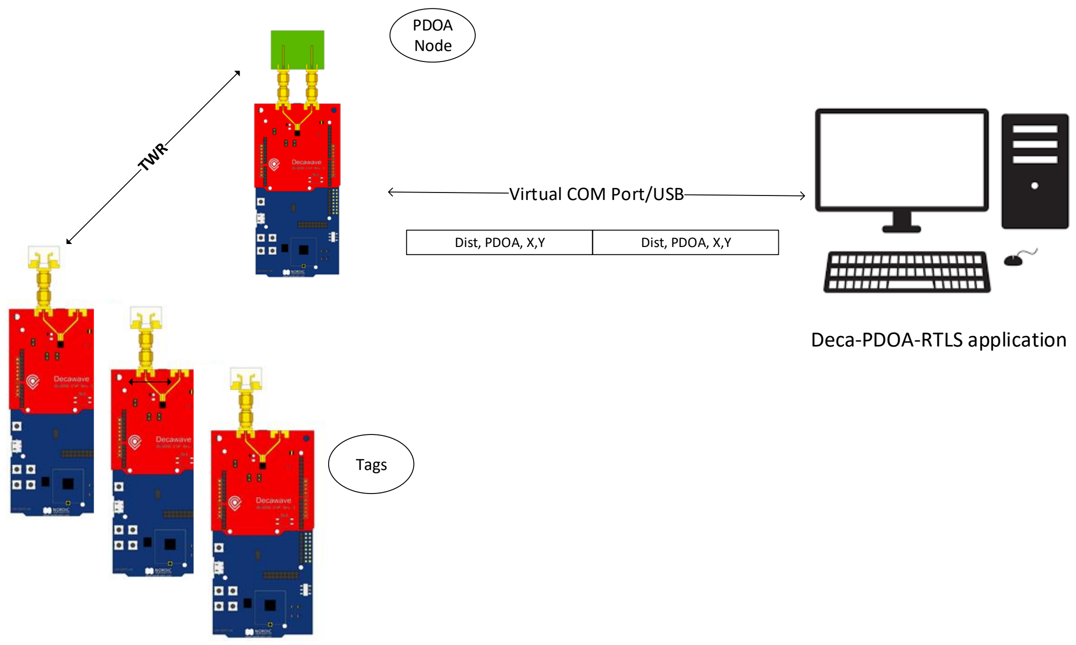


Figure 1: The PDoA Evaluation Kit system components

The Deca-PDoA-RTLS application is built with the Qt framework. Section 3 Qt SDK Installation Guide describes how to set up and build the project.

The *RTLSDisplayApplication class* is the singleton class which handles the application and its main parts are:

- Main window (*_mainWindow* see 2.4) – this is the application's main user interface, it contains:
  - Drop down menus: View and Help
  - The main display area which shows position of tags and the node on a floor plan (*Graphics Widget->scene* see 2.4.1.2).
  - Table with list of known and discovered tags and their configuration (*Graphics Widget->tagTable* see 2.4.1.1)
- Serial connection (*_serialConnection* see 2.2*)* – this is used to establish and monitor Virtual COM port connection with the node over the USB.
- RTLS client (*_client* see 2.3) – this processes the range, PDoA and x,y report messages from Virtual COM port connection, and sends location estimate to the main display (*Graphics Widget)*.
- Configuration (*_viewSettings* see 2.4.4) – used for the general application and floorplan/view configurations

- Loading and Saving osf the tag and display parameters and configurations

These are described in more detail in the following chapters, whereas the Figure 2 gives overview of the main functional blocks. Table 2 gives the list of all the source files and their brief descriptions.

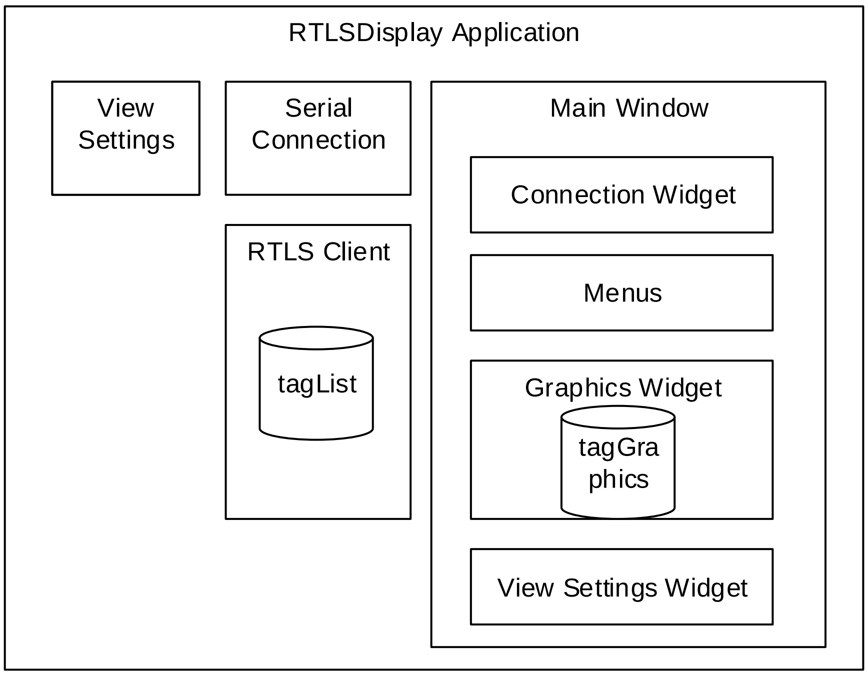


Figure 2: Deca-PDoA-RTLS GUI application - main functional blocks

Table 1: Signals to and from the main functional blocks

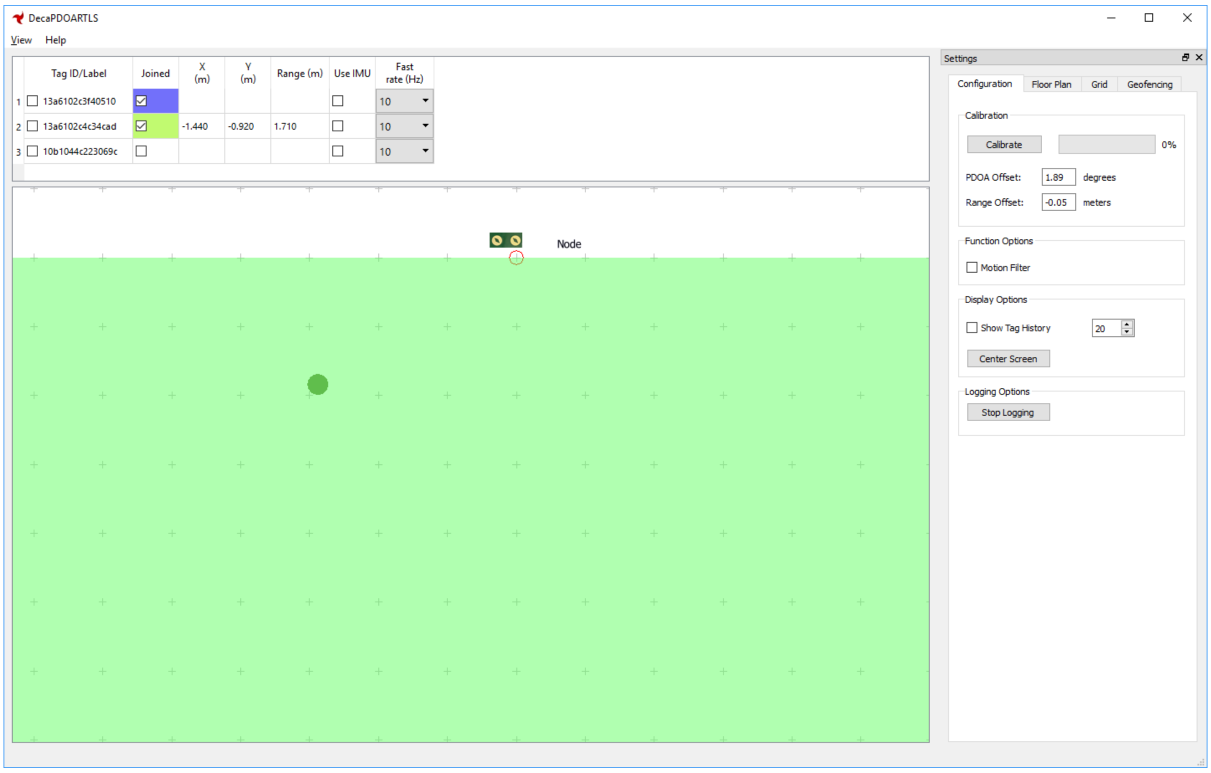
Signal	Brief description
nodePos	RTLS Client to Graphics Widget: to display the PDoA node
tagPos	RTLS Client to Graphics Widget: display the tag
tagRange	RTLS Client to Graphics Widget: update range and x,y in the table
centerOnNodes	RTLS Client to Graphics Widget: fit tags and node in the view
addDiscoveredTag	RTLS Client to Graphics Widget: add tag to table
clearTags	RTLS Client to Graphics Widget: to clear tags table and graphics
statusBarMessage	RTLS Client to Main Window: update text in the status bar
statusBarMessage	Serial Connection to Main Window: update text in the status bar
connectionStateChanged	Serial Connection to Main Window: update change in COM port connection status
serialOpened	Serial Connection to RTLS Client: COM port connection opened
connectionStateChanged	Serial Connection to RTLS Client: update change in COM port connection status
serialOpened	Serial Connection to View Settings Widget: COM port connection opened
connectionStateChanged	Serial Connection to Connection Widget: update change in COM port connection status

Table 2: List of source files in the Deca-PDoA-RTLS GUI application

Filename	Brief description
main.cpp	This is the application main entry point
RTLSDisplayApplication.cpp	The DecaPDoARTLS application constructor – source file
RTLSDisplayApplication.h	The DecaPDoARTLS application constructor – header file
ViewSettings.cpp	View configuration class – source code
ViewSettings.h	View configuration class – header
RTLSClient.cpp	RTLSClient (consumes TOF, PD and X&Y reports) class – source code
RTLSClient.h	RTLSClient (consumes TOF, PD and X&Y reports) class – header
SerialConnection.cpp	Serial (COM port) connection management class – source code
SerialConnection.h	Serial (COM port) connection management class – header
AbstractTool.h	Abstract tool – header
OriginTool.cpp	Origin manipulation tool – source file
OriginTool.h	Origin manipulation tool – header file
GeoFenceTool.c	GeoFencing tool – source file
GeoFenceTool.h	GeoFencing tool – header file
RubberBandTool.cpp	Rubberband tool – source file
RubberBandTool.h	Rubberband tool – header file
ScaleTool.cpp	Scale manipulation tool – source file
ScaleTool.h	Scale manipulation tool – header file
QPropertyModel.cpp	QProperty Model – source code
QPropertyModel.h	QProperty Model – header file
connectionwidget.cpp	Connection widget - source
connectionwidget.h	Connection widget - header
connectionwidget.ui	Connection widget - UI
GraphicsView.cpp	GraphicsView widget - source
GraphicsView.h	GraphicsView widget - header
GraphicsWidget.cpp	GraphicsWidget widget - source
GraphicsWidget.h	GraphicsWidget widget - header
GraphicsWidget.ui	GraphicsWidget widget - UI
mainwindow.cpp	mainwindow widget - source
mainwindow.h	mainwindow widget - header
mainwindow.ui	mainwindow widget - UI
MinimapView.cpp	MinimapView widget - source
MinimapView.h	MinimapView widget - header
ViewSettingsWidget.cpp	ViewSettingsWidget widget - source
ViewSettingsWidget.h	ViewSettingsWidget widget - header
ViewSettingsWidget.ui	ViewSettingsWidget widget - UI
json_utils.h	JSON format parser for TOF, PDoA and X,Y reports - header
json_utils.cpp	JSON format parser for TOF, PDoA and X,Y reports - source

2 DECA-PDOA-RTLS APPLICATION OVERVIEW

This chapter gives an overview about each of the Deca-PDoA-RTLS application functional blocks and UI components as shown in Figure 2. Figure 3 shows the application running with one tag ranging to the PDoA node, and node reporting the tag's X and Y coordinates to it:



		Tag ID/Label	Joined	X (m)	Y (m)	Range (m)	Use IMU	Fast rate (Hz)
1	☐	13a6102c3f40510	☑				☐	10
2	☐	13a6102c4c34cad	☑	-1.440	-0.920	1.710	☐	10
3	☐	10b1044c223069c	☐				☐	10

PDOA Offset: 1.89 degrees
Range Offset: -0.05 meters
☐ Motion Filter
☐ Show Tag History 20

Figure 3: Deca-PDoA-RTLS application – UI

2.1 RTLSDisplayApplication class

The *RTLSDisplayApplication* class is a singleton class which handles the Deca-PDoA-RTLS application. This class is called from main.c. It initialises the application and its parts:

- the _serialConnection is used for managing the COM port connection
- the _client consumes the data received over the COM port connection and sends the calculated location estimation data to the graphical display (graphicsWidget) part of the _mainWindow
- _mainWindow also holds the various other GUI parts: connection widget (_cWidget), statusBar, _viewSettings_w (configuration); dockable widgets: viewSettings_dw and minimap_dw) and menu items (viewMenu, helpMenu),
- the _viewSettings is used to hold many properties about the view, such as the grid and viewplan settings for configuration of the graphical display.

When built in *Release* mode, the application will automatically try to connect to the PDoA node. If the COM port connection fails the user will be prompted to close the application or try again. If the COM port connection is successful, the application will then open the *Main window* (_mainWindow) and finish configuration / initialisation of all of its parts. As part of initialisation the configuration files, if present, will also be loaded.

2.2 Serial Connection class

The *Serial Connection* class is used to establish and monitor the COM port connection to the PDoA node. When the Deca-PDoA-RTLS application is started it will scan all of the COM ports available in the system (PC) and check which ones reply to “deca$”. If the reply contains “Node” string (see function findSerialDevices()) the application will add the COM port to an PDoA Node ports list. I.e. for each found COM port the *Serial Connection* will send “deca$” string and if the returned string matches JSON format:

```
JS0089{"Info":{
"Device":"PDoA Node",
"Version":"4.1.0",
"Build":"Oct 26 2018 11:03:34",
"Driver":"DW1000 Device Driver Version 04.01.01"}}
ok
```

the COM port will be added to the list of PDoA Node ports (_ports).

When the application is built in *Release* mode the *Serial Connection* will then connect to the 1st COM port in the list (see function openConnection()), automatically (without any user interaction). For the connection to be successful the PDoA node needs to be already plugged into the PC, and its port enumerated. Firstly, the *Serial Connection* sends “stop” command, followed by “deca$” command. If it receives the correct JSON Info string, it will pass all future the COM port data processing to the RTLS client.

As part of connection the *Serial Connection* will also start a periodic 30 s timer, which will send “getDlist” command to request a list of any newly discovered tags to the GUI, so that the tag table can be updated. For details of these commands please see PDoA Node Source Code Guide.

The GUI COM port *Connection state* can be: Disconnected, Connecting, Connected or Connection Failed.

2.3 RTLS client class

The *RTLS Client* class is used to process the incoming data on the COM port and send the calculated tag’s location to the graphical display (graphicsWidget). It will also send details of any newly discovered tags to the graphical display to update the tag’s table.

On connection to the PDoA node, the *RTLS client* class will initialise its data structures and open a log file (./Logs/yyyyMMdd_hhmmssPDoARTLS_log.txt). It will then assign the COM port data handler to itself, and from then on, all the data received from the COM port will be handled by the *RTLS client* only. It will also request from the node a list of known tags (“getKlist”), and issue a “node 0” command which will enable PDoA node to range to any known tags.

```
_serial = RTLSDisplayApplication::serialConnection()->serialPort();
connect(_serial, SIGNAL(readyRead()), this, SLOT(newData()));
```

For each range and PD report received (a JSON “TWR” object) the application will extract the report data and calculated X and Y coordinates of the tag’s location. The tag’s location estimate (x and y coordinates) will then be sent to the graphical display (graphicsWidget) to update tag’s position on the application display window. The node is always placed on 0,0 2D-coordinate and the tag’s position will be drawn “under” the node (± 90^o), as shown in Figure 3. The JSON TWR output is as shown:

```
JS006A{"TWR": {"a16":"4096","R":53,"T":5126,"D":112,"P":-
161,"Xcm":112,"Ycm":0,"O":336,"V":0,"X":0,"Y":0,"Z":0}}
```

When the logging is enabled the *RTLS client* will log the serial port data (TOF, PDoA, X and Y coordinates) as well as the calculated positions. The logging can be stopped by pressing the “Stop Logging” button, a new log file can be started by pressing “Start Logging” button on the applications configuration window.

2.3.1 Location Display Function

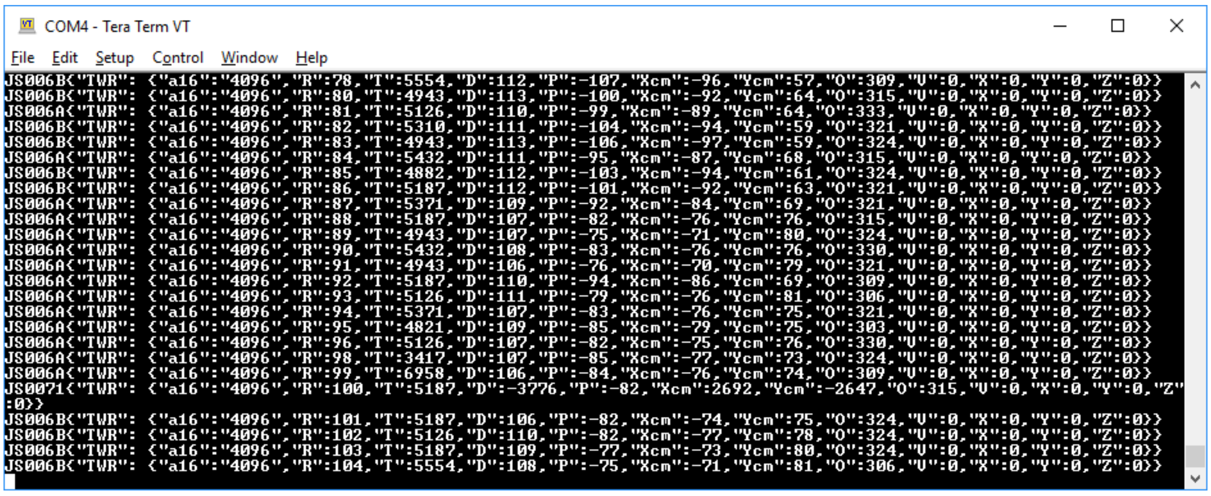
Upon the reception of JSON range and PDoA report, the *RTLS client* extracts tag’s location (x and y coordinates) and sends a signal to the graphicsWidget to updated tag’s location on the display window.

2.3.2 Range and Phase Difference Report Format

The PDoA node sends range, PDoA and other tag data as a JSON “TWR” object over the COM port. The format of which is, as shown in Figure 4:
{ “TWR”:
{ “a16” : “aaaa”, “R” : “rrr”, “T” : “ttt”, “D” : “ddd”, “P” : “ppp”, “Xcm”:xxx, “Ycm”:yyy, “O” : “o”, “V” : “v”, “X” : “x”, “Y” : “y”, “Z” : “z” }
}

The “TWR” object consists of:

aaaa	this is the tag 16-bit address (hexadecimal, 16-bit number)
rrr	this is the range sequence number (decimal, 8-bit number)
ttt	this is the time of range exchange in micro seconds (decimal, 32-bit number)
ddd	this is the tag to node range (in cm, integer)
ppp	this is the phase difference (in degrees, integer)
xxx	this is the X coordinate (in cm, integer)
yyy	this is the Y coordinate (in cm, integer)
o	this is the clock offset in ppm*10^2
v	this indicates if tag is stationary (LSB is 1 when stationary)
x, y, z	these are IMU x, y, z raw data



```
JS006B{"TWR": {"a16":"4096","R":78,"T":5554,"D":112,"P":-107,"Xcm":-96,"Ycm":57,"O":309,"V":0,"X":0,"Y":0,"Z":0}}
JS006B{"TWR": {"a16":"4096","R":80,"T":4943,"D":113,"P":-100,"Xcm":-92,"Ycm":64,"O":315,"V":0,"X":0,"Y":0,"Z":0}}
JS006A{"TWR": {"a16":"4096","R":81,"T":5126,"D":110,"P":-99,"Xcm":-89,"Ycm":64,"O":333,"V":0,"X":0,"Y":0,"Z":0}}
JS006B{"TWR": {"a16":"4096","R":82,"T":5310,"D":111,"P":-104,"Xcm":-94,"Ycm":59,"O":321,"V":0,"X":0,"Y":0,"Z":0}}
JS006B{"TWR": {"a16":"4096","R":83,"T":4943,"D":113,"P":-106,"Xcm":-97,"Ycm":59,"O":324,"V":0,"X":0,"Y":0,"Z":0}}
JS006A{"TWR": {"a16":"4096","R":84,"T":5432,"D":111,"P":-95,"Xcm":-87,"Ycm":68,"O":315,"V":0,"X":0,"Y":0,"Z":0}}
JS006B{"TWR": {"a16":"4096","R":85,"T":4882,"D":112,"P":-103,"Xcm":-94,"Ycm":61,"O":324,"V":0,"X":0,"Y":0,"Z":0}}
JS006B{"TWR": {"a16":"4096","R":86,"T":5187,"D":112,"P":-101,"Xcm":-92,"Ycm":63,"O":321,"V":0,"X":0,"Y":0,"Z":0}}
JS006A{"TWR": {"a16":"4096","R":87,"T":5371,"D":109,"P":-92,"Xcm":-84,"Ycm":69,"O":321,"V":0,"X":0,"Y":0,"Z":0}}
JS006A{"TWR": {"a16":"4096","R":88,"T":5187,"D":107,"P":-82,"Xcm":-76,"Ycm":76,"O":315,"V":0,"X":0,"Y":0,"Z":0}}
JS006A{"TWR": {"a16":"4096","R":89,"T":4943,"D":107,"P":-75,"Xcm":-71,"Ycm":80,"O":324,"V":0,"X":0,"Y":0,"Z":0}}
JS006A{"TWR": {"a16":"4096","R":90,"T":5432,"D":108,"P":-83,"Xcm":-76,"Ycm":76,"O":330,"V":0,"X":0,"Y":0,"Z":0}}
JS006A{"TWR": {"a16":"4096","R":91,"T":4943,"D":106,"P":-76,"Xcm":-70,"Ycm":79,"O":321,"V":0,"X":0,"Y":0,"Z":0}}
JS006A{"TWR": {"a16":"4096","R":92,"T":5187,"D":110,"P":-94,"Xcm":-86,"Ycm":69,"O":309,"V":0,"X":0,"Y":0,"Z":0}}
JS006A{"TWR": {"a16":"4096","R":93,"T":5126,"D":111,"P":-79,"Xcm":-76,"Ycm":81,"O":306,"V":0,"X":0,"Y":0,"Z":0}}
JS006A{"TWR": {"a16":"4096","R":94,"T":5371,"D":107,"P":-83,"Xcm":-76,"Ycm":75,"O":321,"V":0,"X":0,"Y":0,"Z":0}}
JS006A{"TWR": {"a16":"4096","R":95,"T":4821,"D":109,"P":-85,"Xcm":-79,"Ycm":75,"O":303,"V":0,"X":0,"Y":0,"Z":0}}
JS006A{"TWR": {"a16":"4096","R":96,"T":5126,"D":107,"P":-82,"Xcm":-75,"Ycm":76,"O":330,"V":0,"X":0,"Y":0,"Z":0}}
JS006A{"TWR": {"a16":"4096","R":98,"T":3417,"D":107,"P":-85,"Xcm":-77,"Ycm":73,"O":324,"V":0,"X":0,"Y":0,"Z":0}}
JS006A{"TWR": {"a16":"4096","R":99,"T":6958,"D":106,"P":-84,"Xcm":-76,"Ycm":74,"O":309,"V":0,"X":0,"Y":0,"Z":0}}
JS0071{"TWR": {"a16":"4096","R":100,"T":5187,"D":-3776,"P":-82,"Xcm":2692,"Ycm":-2647,"O":315,"V":0,"X":0,"Y":0,"Z"
:0}}
JS006B{"TWR": {"a16":"4096","R":101,"T":5187,"D":106,"P":-82,"Xcm":-74,"Ycm":75,"O":324,"V":0,"X":0,"Y":0,"Z":0}}
JS006B{"TWR": {"a16":"4096","R":102,"T":5126,"D":110,"P":-82,"Xcm":-77,"Ycm":78,"O":324,"V":0,"X":0,"Y":0,"Z":0}}
JS006B{"TWR": {"a16":"4096","R":103,"T":5187,"D":109,"P":-77,"Xcm":-73,"Ycm":80,"O":324,"V":0,"X":0,"Y":0,"Z":0}}
JS006B{"TWR": {"a16":"4096","R":104,"T":5554,"D":108,"P":-75,"Xcm":-71,"Ycm":81,"O":306,"V":0,"X":0,"Y":0,"Z":0}}
```

Figure 4: Example Teraterm window showing the JSON "TWR" info sent via COM port

2.3.3 Tag Discovery Report Format

The PDoA node will output a list of tags which are part of its network and also any newly discovered tags as a reply to "getKlist" and "getDlist" commands. (For more information on this please see the PDoA Node Source Code Guide [1]).

```
{"DList": ["addr64_t1","addr64_t2","addr64_t3"]}

{"KList": [
        { "slot" : "1", "addr64" : "t0 64-bit", "addr16" : "t0 16-bit", "multFast" : "1", "multSlow" : "2",
"mode" : "1"},
        { "slot" : "2", "addr64" : "t1 64-bit", "addr16" : "t2 16-bit", "multFast" : "1", "multSlow" : "2",
"mode" : "1"},
        ]}
```

The *RTLS client* will parse the JSON tag data and update its tag array structures and send the tag information to the graphicsWidget to update the tag information in the tag table.

2.4 Main window

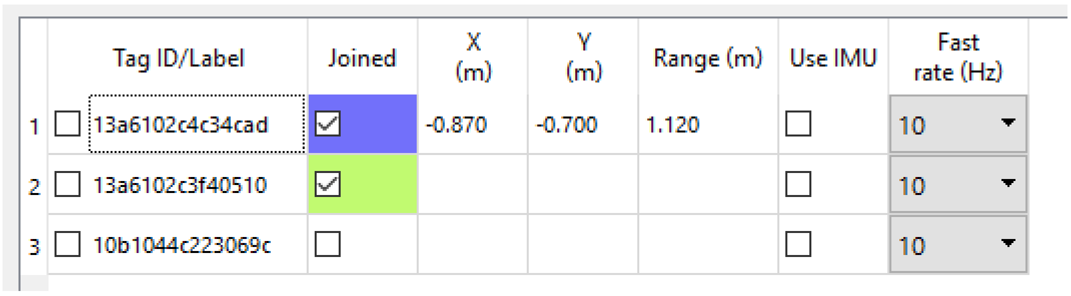
The *Main window* is a class that contains the drop down menus, main display area, tag information table and other information and control widgets as described in the sections below:

2.4.1 Graphics

The graphical display (graphicsWidget) part of the _mainWindow which holds the various GUI parts namely these are Tag and PDoA node table pane, the display pane. The display pane consists of *Graphics View* which has visible rectangle to keep track of the visible rectangle 2.5.1 and tools 2.6 which operate on it and the *Grid Layout*.

2.4.1.1 Tag Table

The *Tag* table is a Qt Table widget, part of the graphicsWidget. It contains Tag's ID, position information and range measurements, as shown in Figure 5.



		Tag ID/Label	Joined	X (m)	Y (m)	Range (m)	Use IMU	Fast rate (Hz)
1	☐	13a6102c4c34cad	✓	-0.870	-0.700	1.120	☐	10
2	☐	13a6102c3f40510	✓				☐	10
3	☐	10b1044c223069c	☐				☐	10

Figure 5: Tag table

- Tag ID/Label:
    - The checkbox is used to display the label beside the tag in the display panel
    - The tag label can be changed by double-clicking on the label name and editing the label field. The default name is the 64-bit ID of the tag
- Joined:
    - Check this to enable this tag to join the network of the current node
- X (m)
    - Horizontal position from the node
- Y (m)
    - Vertical position from the node
- Range (m)
    - Displays the range from this tag to the node
- Use IMU
    - If checked, the accelerometer is used to detect stationary or non-stationary mode.
    - If stationary, the circle of the tag on the display changes to an empty circle. The location rate drops down to 0.1 Hz
    - If non-stationary, the location rate reverts to the fast rate that is selected in the pull-down menu
- Fast rate (Hz)
    - The default location rate to use when moving, or if the 'Use IMU' is not checked, when stationary too

2.4.1.2 Display pane

The display pane has a _scene (QGraphicsScene) element, it provides a surface for managing a large number of 2D graphical items. It is the _scene part of *Graphics View* 2.5.1. When the graphical display widget (graphicsWidget) receives data from the *RTLS client* it will:

- Add/update tag's new visual position
- It will remove tag's old positions
- Update labels

2.4.2 Connection widget

The *Connection widget* contains the *Serial Connection* (COM port) configuration options. This widget is hidden from the user (when the application is compiled in *Release* mode) and the RTLS application connects automatically to 1st found PDoA node COM port.

2.4.3 Status bar

The *Status bar* is used to display status messages/strings:

- "DecaPDoARTLS Node connected (*version number*)" – on COM port open it will display the version number of the node the application is connected to.
- "Open error" – if the COM port did not open successfully.

2.4.4 View settings

The *View settings* widget contains three Tabs: Configuration, Floor Plan and Grid.

2.4.4.1 Configuration tab

The Configuration tab contains three groups of application configuration options and settings:

1. Phase Calibration: this is used to calibrate the 0^o phase offset.
2. Function Options: here user can select filtering option
3. Display Options: here the user can select to display the tag history. Use full screen to display PDoA node and the tag, and also enabled or disable logging.

2.4.4.2 Floor Plan tab

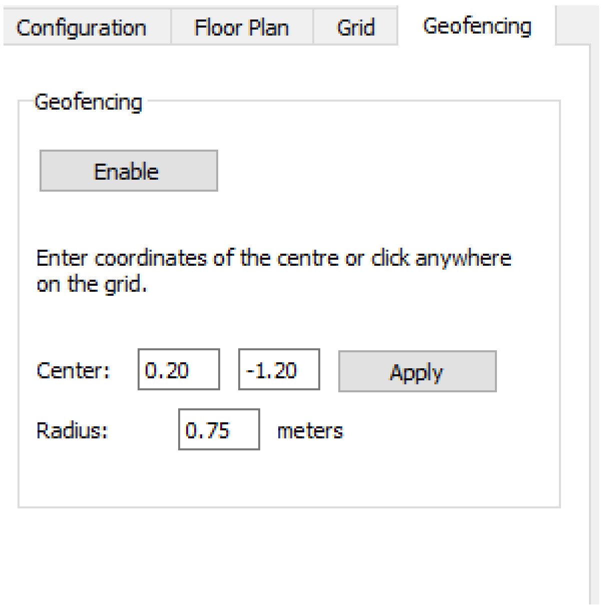
The Floor Plan tab contains the floor plan image loading and configuration options as shown in Figure 8 and described in 2.5.4.

2.4.4.3 Grid tab

The Grid tab contains the grid configuration options as shown in Figure 8 and described in 2.5.4.

2.4.4.4 Geofencing tab

The Geofencing tab contains the geofencing configuration options as shown in Figure 6Figure 8 and described in 2.5.4. The geofencing area is shown in Figure 7.



Enter coordinates of the centre or click anywhere on the grid.

Center: 0.20 -1.20

Radius: 0.75 meters

Figure 6: Geofencing tag configuration options

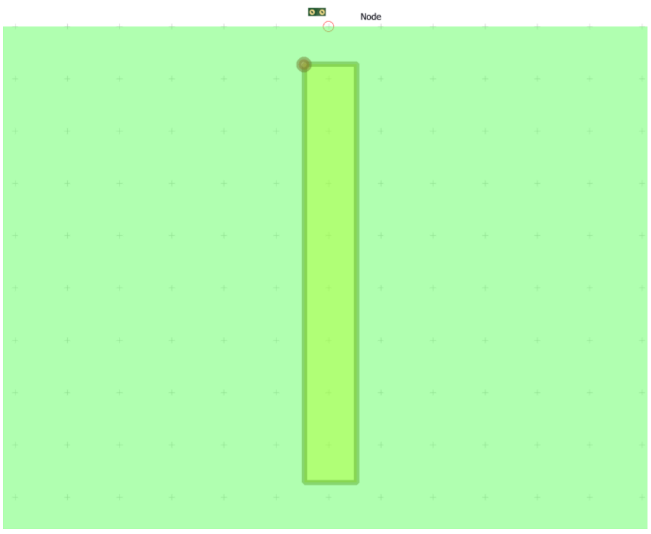


Figure 7: Example geofencing area

2.4.5 Minimap view

The *Minimap view* widget allows the user, by using the mouse, to select different regions of the floorplan to be displayed in the main scene. It is only operational if a floorplan has been loaded. Š

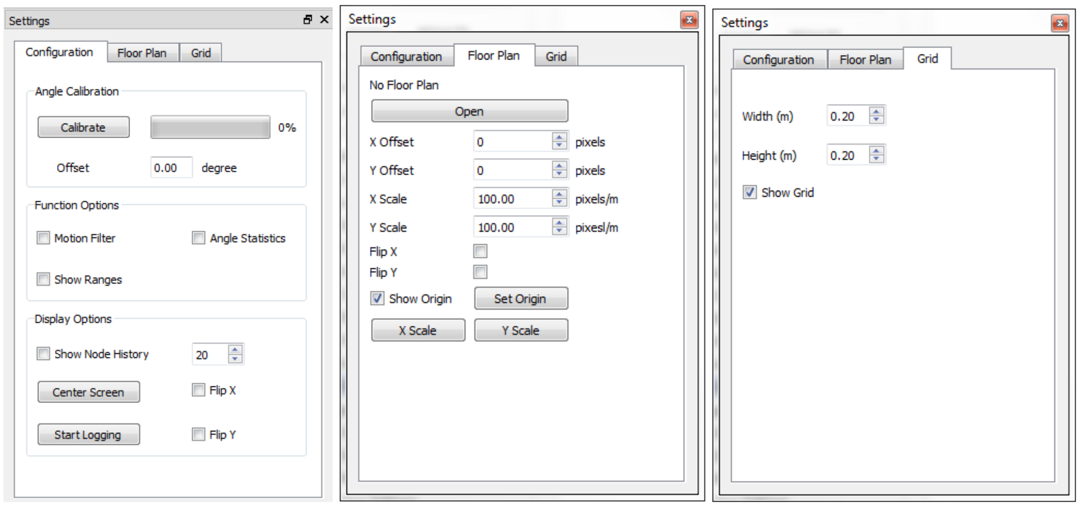
2.4.6 Drop down menus

There are two drop down menus: View and Help. These are described below:

2.4.6.1 VIEW

View menu contains the following items: -

- ***View Settings*** – selecting this will open a view configuration widget. This allows the user to upload a floor plan and specify the grid, X and Y axis scale and origin positions: -



Settings – Configuration tab

Angle Calibration
Calibrate 0%
Offset 0.00 degree

Function Options
☐ Motion Filter ☐ Angle Statistics
☐ Show Ranges

Display Options
☐ Show Node History 20
☐ Flip X
☐ Flip Y

Settings – Floor Plan tab

No Floor Plan
X Offset 0 pixels
Y Offset 0 pixels
X Scale 100.00 pixels/m
Y Scale 100.00 pixesl/m
Flip X ☐
Flip Y ☐
☑ Show Origin

Settings – Grid tab

Width (m) 0.20
Height (m) 0.20
☑ Show Grid

Figure 8: View settings widget

  - *Configuration* tab:

Field Name	Description
Calibrate	This enables calibration of the 0^o Phase Difference (PD)
Offset	This is the 0^o PD offset (can be entered here if already known)
Motion Filter	Enable/disable motion filter
Center Screen	Places all of the nodes on the screen (zooms out)
Start Logging	Logging can be enabled or disabled
Show History	Shows previous tag positions (moving history of tag positions)

- *Grid* tab:

Field Name	Description
Width	Sets the horizontal grid spacing, unit is meters
Height	Sets the vertical grid spacing, unit is meters
show	Shows or hides the grid

- *Floor Plan* tab:

Field Name	Description
Open	This lets the user upload an image of the floor plan of the area where the node are installed
X offset	This is the origin offset in the X direction from the 0, 0 point on the floorplan (in pixels).
Y offset	This is the origin offset in the Y direction from the 0, 0 point on the floorplan (in pixels)
X scale	This is the scale (pixels per meter) of the x axis (in pixels per meter)
Y scale	This is the scale (pixels per meter) of the y axis (in pixels per meter)
Flip X	This flips the image in the x-axis
Flip Y	This flips the image in the y-axis
show	This shows or hides the origin in the map
Set Origin	This button lets the user click and set the origin coordinates
X Scale button	Pressing this button produces a measuring tool with which the user can firstly select a distance on the map and then enter the actual distance (in meters) that range corresponds to, this sets the *X scale* value
Y Scale button	Pressing this button produces a measuring tool with which the user can firstly select a distance on the map and then enter the actual distance (in meters) that range corresponds to, this sets the *Y scale* value

- *Geofencing* tab:

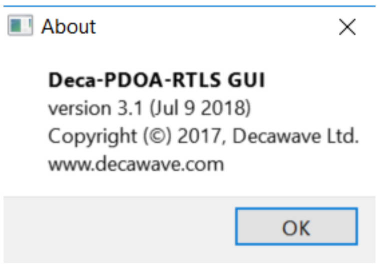
Field Name	Description
Enable/Disable	Enable/Disable – turn ON/OFF the geofencing feature. Disabling will also hide

Field Name	Description
button	the geofencing area
Center	Sets the center point of the geofencing area
Radius	Sets the radius of geofencing area
Apply button	Applies any changes to center position.

- ***Minimap*** – selecting this opens a Minimap widget, which shows the loaded image and the zoomed in area (which is shown in the Main Display Area window).

2.4.6.2 HELP

- – this opens the "About" window which provides information on the revision of the client.


About

Deca-PDOA-RTLS GUI
version 3.1 (Jul 9 2018)
Copyright (©) 2017, Decawave Ltd.
www.decawave.com

Figure 9: About window

2.5 Widgets and Views

The Deca-PDoA-RTLS application has a number of Widgets and View classes they are used to display the data, and allow user to change/configure the various parameters.

2.5.1 Graphics View

The *Graphics View* class draws the scene and provides user interaction using the mouse. User interaction can be complex as we have to handle many different interactions based on very few mouse events:

- Zooming using the scroll wheel
- Panning by dragging
- Contextual menu by right-click
- Cancelling the current tool by right-click
- Starting a *Rubber Band Tool* on Ctrl+Drag (2.6.4)
- Tools listening to AbstractTool::clicked() events (2.6.1)
- Tools listening to AbstractTool::mousePressEvent() events (2.6.1)

Most of them are handled by the mousePressEvent()/mouseMoveEvent()/mouseReleaseEvent() cycle. During mousePressEvent(), we find the suited action, and decide of a MouseContext based on that. The MouseContext is kept until mouseReleaseEvent()

The GraphicsView keeps track of the visible rectangle, in scene coordinates. This rectangle will always be visible on the screen. Initially, the visible rectangle is the square from (0, 0) to (1, 1). The visible rectangle can be transformed using translateView() or scaleView() or changed using setVisibleRect(). Whenever the visible rectangle changes, for any reason, the visibleRectChanged()

signal is called.
Tools allow simple interaction inside the scene. A new tool can be set using setTool(). The tool then remains active until it's AbstractTool::done() signal is emitted. When ESC button or right click is pressed, the view attempts to cancel the tool by calling AbstractTool::cancel().

2.5.2 Graphics

The *Graphics Widget* is used to display tags and the PDoA node on the main display. It adds tag and node items to the scene, and updates the x and y co-ordinates as it gets new data from the *RTLS Client*.

The list of tags is also shown with their configuration properties, here the user can add a new tag label, change the IMU use or fast and slow location rates.

2.5.3 Minimap

The *Minimap* view shows a zoomed out view of the whole floorplan. It allows the user to quickly select and zoom into a particular place on the floor plan loaded.

2.5.4 View Settings

The *View Settings* widget allows the user to configure the grid size, floorplan settings etc. This is explained in detail in section 2.4.6.1 View Settings.

2.6 Tools and Utilities

2.6.1 Abstract Tool

The Abstract Tool class is a base class any tool implementations should inherit. Tools allow simple, temporary user interaction in the graphics view, based on mouse events. They are used to set the scale (*Scale Tool 2.6.5*) and origin (*Origin Tool 2.6.2*) of the bitmap, and select nodes using a click and drag style rubber band (*Rubber Band Tool 2.6.4*).

Once a tool is enabled, using GraphicsView::setTool(), the cursor changes to the return value of cursor(), the draw() function gets called when drawing the foreground of the scene, and the tool starts receiving mouse events.

The tool can receive the mouse events through two means, pressing and releasing of the left mouse button. The clicked() function gets called when the left button is pressed and released. For more complex interaction, the mousePressEvent()/mouseMoveEvent()/mouseReleaseEvent() functions can be overridden. When a mouse button is pressed, mousePressEvent is called. If it returns true, then tool grabs the mouse interaction, and will receive mouse events until the button is released. Otherwise the scene will be handling mouse events, and mouseMoveEvent()/mouseReleaseEvent() won't be called.

Note that the two mouse event mechanisms are incompatible. If mousePressEvent returns true, clicked() will not be called for this mouse click.

2.6.2 Origin Tool

The *Origin Tool* class is a tool used for setting the floorplan's origin point. It reacts to the clicked() event. When clicked() is called, the origin is calculated based on the click position, and the tool finishes right away.

2.6.3 GeoFence Tool

The *GeoFence Tool* class is a tool used for setting the geofencing area origin point. It reacts to the clicked() event. When clicked() is called, the centre is calculated based on the click position, and the tool finishes right away.

2.6.4 Rubber Band Tool

The *Rubber Band Tool* class is a tool used to select nodes using a click and drag to draw a selection box. Once the user has started the drag, a rectangular selection box is drawn, and all items within the box get selected. In order to differentiate this tool's click and drag with the one used to move the scene, the tool is started if the Control button is held during the initial mouse press event.

This is handled by the GraphicsView::mousePressEvent(), and is outside the scope of this class.

2.6.5 Scale Tool

The *Scale Tool* class is a tool used to change the floorplan's scale. It allows the user to select two points by waiting for two consecutive clicked() events. Once the user has clicked on two points, the tool shows a popup to enter the distance between the two points. The scale is recalculated and stored.

2.6.6 QProperty Model

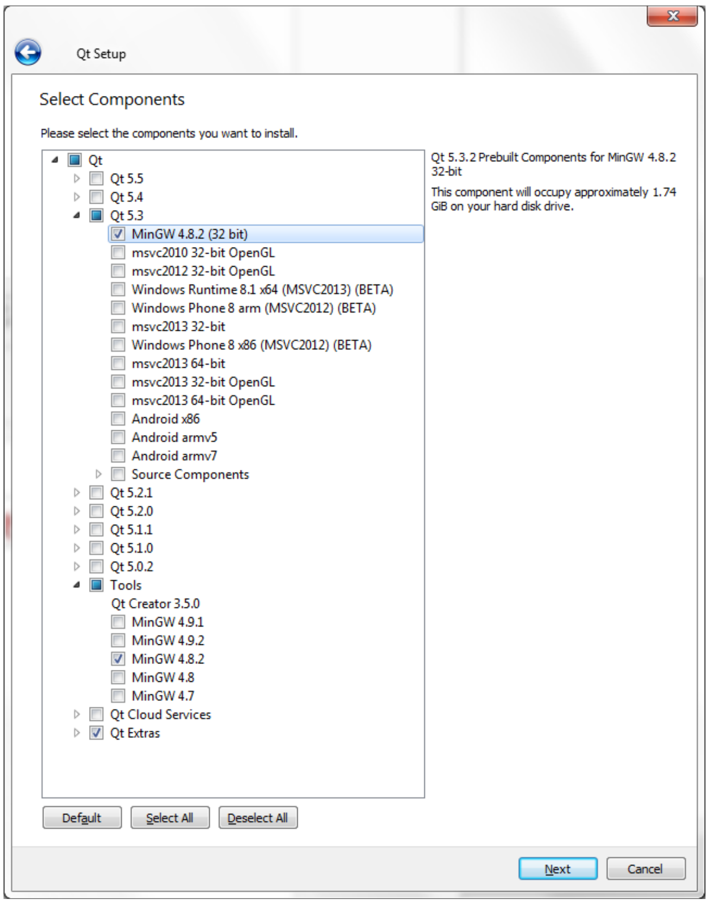
The *QProperty Model* is a class for turning any QObject-derived subclass with properties into a one-row model. (Copyright 2013 - Harvey Chapman hchapman@3gfp.com Source: https://gist.github.com/sr105/7955969). QPropertyModel creates a single row data model consisting of columns mapping to properties in a QObject. The column list can be retrieved as a QStringList, and a method exists to convert the property names to column numbers.

3 QT SDK INSTALLATION GUIDE

The Deca-PDoA-RTLS application is built under the Qt Framework, the released binary is compiled and built with Qt Creator (4.2.2) based on Qt 5.8.0 (MSVC 2015, 32bit). The Qt installer can be found online: http://www.qt.io/download-open-source/. Select the "Qt Online Installer" for your platform, download and build the application.

Once the installer is downloaded, run it, specify the installation path and select the following components:

- Qt -> Qt 5.3 -> MinGW 4.8.2
- Qt -> Tools -> Qt Creator
- Qt -> Tools -> MinGW 4.8.2


Qt Setup

Select Components

Please select the components you want to install.

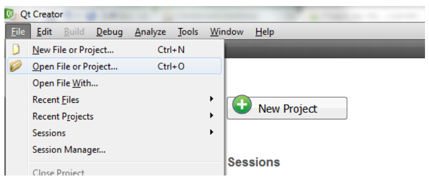
- ▣ Qt
  - ☐ Qt 5.5
  - ☐ Qt 5.4
  - ▣ Qt 5.3
    - ✓ MinGW 4.8.2 (32 bit)
    - ☐ msvc2010 32-bit OpenGL
    - ☐ msvc2012 32-bit OpenGL
    - ☐ Windows Runtime 8.1 x64 (MSVC2013) (BETA)
    - ☐ Windows Phone 8 arm (MSVC2012) (BETA)
    - ☐ msvc2013 32-bit
    - ☐ Windows Phone 8 x86 (MSVC2012) (BETA)
    - ☐ msvc2013 64-bit
    - ☐ msvc2013 32-bit OpenGL
    - ☐ msvc2013 64-bit OpenGL
    - ☐ Android x86
    - ☐ Android armv5
    - ☐ Android armv7
    - ☐ Source Components
  - ☐ Qt 5.2.1
  - ☐ Qt 5.2.0
  - ☐ Qt 5.1.1
  - ☐ Qt 5.1.0
  - ☐ Qt 5.0.2
  - ▣ Tools
    - Qt Creator 3.5.0
    - ☐ MinGW 4.9.1
    - ☐ MinGW 4.9.2
    - ✓ MinGW 4.8.2
    - ☐ MinGW 4.8
    - ☐ MinGW 4.7
  - ☐ Qt Cloud Services
  - ✓ Qt Extras

Qt 5.3.2 Prebuilt Components for MinGW 4.8.2 32-bit

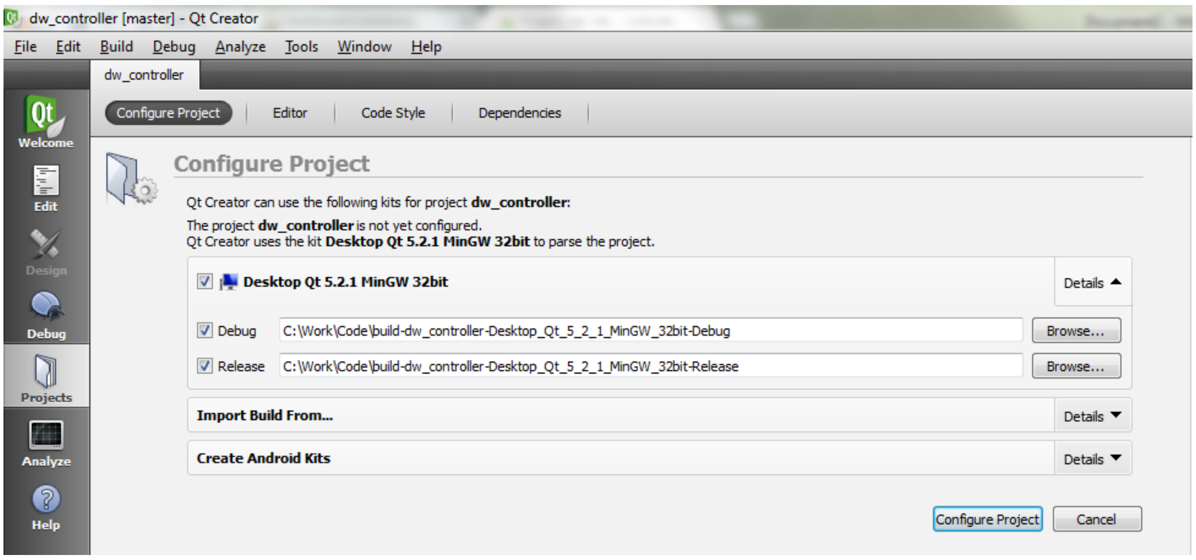
This component will occupy approximately 1.74 GiB on your hard disk drive.

Choose the default options until the end of the wizard.

Run the Qt Creator software. Under the *File* menu, select *Open File or Project*



Select the *.pro* file inside the Project's directory. The first time the project is opened, it must be configured. Leave the default options, and click *Configure Project*



Configure Project

Qt Creator can use the following kits for project **dw_controller**:

The project **dw_controller** is not yet configured.
Qt Creator uses the kit **Desktop Qt 5.2.1 MinGW 32bit** to parse the project.

[✓] **Desktop Qt 5.2.1 MinGW 32bit** Details

[✓] Debug C:\Work\Code\build-dw_controller-Desktop_Qt_5_2_1_MinGW_32bit-Debug Browse...

[✓] Release C:\Work\Code\build-dw_controller-Desktop_Qt_5_2_1_MinGW_32bit-Release Browse...

Import Build From... Details

Create Android Kits Details

Configure Project Cancel

4 BIBLIOGRAPHY

Table 3: Table of References

Ref	Author	Title
[1]	Decawave	PDoA Node Source Code Guide
[2]	Decawave	PDoA Tag Source Code Guide

5 DOCUMENT HISTORY

Table 4: Document History

Revision	Date	Description
1.0	25th of Oct 2018	Initial release 1.0

6 FURTHER INFORMATION

Decawave develops semiconductors solutions, software, modules, reference designs - that enable real-time, ultra-accurate, ultra-reliable local area micro-location services. Decawave's technology enables an entirely new class of easy to implement, highly secure, intelligent location functionality and services for IoT and smart consumer products and applications.
For further information on this or any other Decawave product, please refer to our website www.decawave.com.