

ARIA Radar Development Kit User Manual

1 Document Version

Version	Date	Description
1.0	2024/09/04	Issued

2 Introduction

ARIA RDK (*Radar Development Kit*) is a complete suite to evaluate ARIA Radar Devices and/or to develop custom application algorithms to radar data. ARIA RDK allows for the simultaneous data acquisition from one or more ARIA radar devices. Consequently, users may exploit ARIA RDK to develop different applications, ranging from single-device evaluation, to customer's application development, and to advanced applications such as radar tomography.

3 Software Architecture

The system architecture is shown in Fig.1.

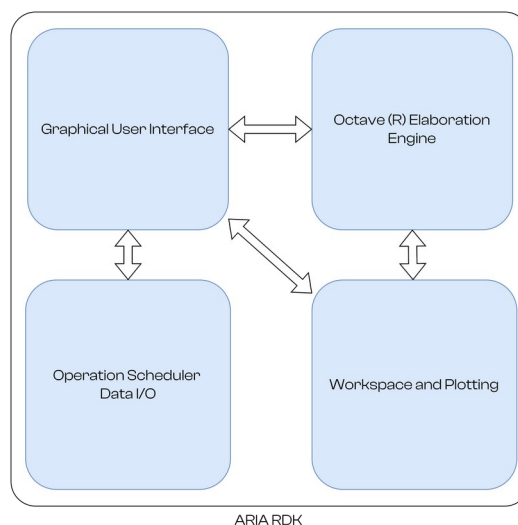


Fig. 1: ARIA RDK Main Architecture

ARIA RDK includes four main components:

1. A Graphical User Interface (Windows and Linux) to create custom projects
2. An interface component to the Scientific Programming Language Octave to run custom algorithms on radar data
3. A Scheduler & Data I/O component to coordinate radar acquisitions and execute user scripts
4. A data workspace management and plotting component

4 Prerequisites

ARIA RDK requires the following components to be compiled and installed in your system:

- Qt6 ($\geq 6.7.2$)
- VTK: The Visualization Toolkit (<https://vtk.org/>)
- JKQTPlotter: an extensive Qt5 & Qt6 Plotter Framework
(<https://jkriege2.github.io/JKQtPlotter/index.html>)
- Eigen: C++ template library for linear algebra ($\geq 3.4.0$) (<https://eigen.tuxfamily.org/>)
- Octave ($\geq 9.2.0$): a scientific programming language environment
- ARIA UWB Toolbox (https://github.com/ariasensing/ARIA_uwb_toolbox.git)
- QWT Plot (<https://qwt.sourceforge.io/index.html>)
- Qscintilla (<https://qscintilla.com/>)

Please refer to respective build/install instruction.

ARIA-RDK is designed to be compiled with CMake®. Currently it has been compiled and tested under Clear Linux SO.

4.1 Building Instructions

The following instructions have been tested in Clear Linux SO, but it is expected full compatibility with other operating systems.

4.1.1 Qt

Install Qt through Qt-Creator self-installer program.

emPulse® is an *ARIA Sensing* trademark.

4.1.2 JKQTPlotter

- Download source code:

```
> git clone https://github.com/jkriege2/JKQtPlotter.git  
> cd [folder where JKQTPlotter is downloaded]  
> mkdir build  
> cd build
```

- Start cmake-gui :

```
>cmake-gui ..
```

- Select the target compiler:

Start Configure and check that Qt6 is found:

Name	Value
QT_DIR	/usr/lib64/cmake/Qt6
Qt6CoreTools_DIR	/usr/lib64/cmake/Qt6CoreTools
Qt6Core_DIR	/usr/lib64/cmake/Qt6Core
Qt6DBusTools_DIR	/usr/lib64/cmake/Qt6DBusTools
Qt6DBus_DIR	/usr/lib64/cmake/Qt6DBus
Qt6GuiTools_DIR	/usr/lib64/cmake/Qt6GuiTools
Qt6Gui_DIR	/usr/lib64/cmake/Qt6Gui
Qt6OpenGLWidgets_DIR	/usr/lib64/cmake/Qt6OpenGLWidgets
Qt6OpenGL_DIR	/usr/lib64/cmake/Qt6OpenGL
Qt6PrintSupport_DIR	/usr/lib64/cmake/Qt6PrintSupport
Qt6Svg_DIR	/usr/lib64/cmake/Qt6Svg
Qt6Test_DIR	/usr/lib64/cmake/Qt6Test
Qt6WidgetsTools_DIR	/usr/lib64/cmake/Qt6WidgetsTools
Qt6Widgets_DIR	/usr/lib64/cmake/Qt6Widgets
Qt6Xml_DIR	/usr/lib64/cmake/Qt6Xml
Qt6_DIR	/usr/lib64/cmake/Qt6

- Verify that "CMAKE_BUILD_TYPE" is set to "Release"
- Set a proper installation folder, the installation folder has to be part of, or included into, OS search path.

BUILD_SHARED_LIBS	<input checked="" type="checkbox"/>
CMAKE_BUILD_TYPE	Release
CMAKE_INSTALL_PREFIX	/usr/local
JKQtPlotter_BUILD_WITH_PRECOMPILED_HEADERS	<input checked="" type="checkbox"/>

- Run *Configure* and *Generate* and build & install:

```
> cmake --build .  
> sudo cmake --install .
```

NOTE: during compilation of ARIA-SDK, PrintSupport may not be found. In this case, the compilation is not successful.

To overcome this, check the following "JKQtPlotter_BUILD_FORCE_NO_PRINTER_SUPPORT"

Name	Value
JKQtPlotter_BUILD_FORCE_NO_PRINTER_SUPPORT	<input type="checkbox"/>
Qt6PrintSupport_DIR	/usr/lib64/cmake/Qt6PrintSupport

4.1.3 VTK

- Download source archive from <https://vtk.org/download/>.
- (Linux) Install the following packages

```
build-essential \  
cmake-curses-gui \  
mesa-common-dev \  
mesa-utils \  
freeglut3-dev \
```



- Create a folder (e.g. ~/VTK) and unarchive the source code.
- Enter the new folder and mkdir build folder. Enter build folder

```
> cd [folder where VTK source code folder is extracted]
> mkdir build
> cd build
```

- Run the configuration cmake

```
> cmake -S path_to_vtk_source -B path_to_vtk_source/VTK-build
```

- Verify / set the following parameters

```
Qt6CoreTools_DIR      /usr/lib64/cmake/Qt6CoreTools
Qt6Core_DIR           /usr/lib64/cmake/Qt6Core
Qt6DBusTools_DIR      /usr/lib64/cmake/Qt6DBusTools
Qt6DBus_DIR           /usr/lib64/cmake/Qt6DBus
Qt6GuiTools_DIR       /usr/lib64/cmake/Qt6GuiTools
Qt6Gui_DIR            /usr/lib64/cmake/Qt6Gui
Qt6OpenGLWidgets_DIR  /usr/lib64/cmake/Qt6OpenGLWidgets
Qt6OpenGL_DIR         /usr/lib64/cmake/Qt6OpenGL
Qt6Sql_DIR            /usr/lib64/cmake/Qt6Sql
Qt6WidgetsTools_DIR   /usr/lib64/cmake/Qt6WidgetsTools
Qt6Widgets_DIR        /usr/lib64/cmake/Qt6Widgets
Qt6_DIR               /usr/lib64/cmake/Qt6
VTK_BUILD_DOCUMENTATION OFF
VTK_BUILD_EXAMPLES    OFF
VTK_BUILD_SCALED_SOA_ARRAYS OFF
VTK_BUILD_SPHINX_DOCUMENTATION OFF
VTK_BUILD_TESTING     OFF
VTK_EXTRA_COMPILER_WARNINGS OFF
VTK_GROUP_ENABLE_Imaging WANT
VTK_GROUP_ENABLE_MPI  DONT_WANT
VTK_GROUP_ENABLE_Qt    YES
VTK_GROUP_ENABLE_Rendering WANT
VTK_GROUP_ENABLE_StandAlone DONT_WANT
VTK_GROUP_ENABLE_Views  WANT
VTK_GROUP_ENABLE_Web    DONT_WANT
VTK_QT_VERSION         6
VTK_SMP_IMPLEMENTATION_TYPE OpenMP
VTK_USE_CUDA           OFF
VTK_USE_LARGE_DATA     OFF
```

(NB Actual Qt paths may be different)

- Verify that the installation prefix is set to a folder included in the search path, e.g.

```
BUILD_SHARED_LIBS      ON
CMAKE_BUILD_TYPE       Release
CMAKE_INSTALL_PREFIX    /usr/local
```

- Press **c** to configure and **g** to generate
- Go into VTK source folder

```
> cmake --build path_to_vtk_source/VTK-build
```

```
> sudo cmake --install . path_to_vtk_source/VTK-build
```

4.1.4 ARIA UWB Toolbox

The ARIA UWB Toolbox is an Octave package required to allow for the Octave scripts to inquiry, set and control module parameters at any desired code line.

Download it in a temporary folder and compress it (e.g. *aria_uwb_toolbox.tar.gz*).

From the Octave command line, move to the folder containing the archived toolbox.

Type:

```
>> pkg install aria_uwb_toolbox.tar.gz
```

When done (it may take few minutes), type

```
>> pkg load aria_uwb_toolbox
```

Note: ARIA UWB Toolbox requires two octave packages: **signal** and **communications**. You may install those by typing:

```
>> pkg install --forge signal  
>> pkg install --forge communications
```

Please ensure to have an available internet connection when installing those extra packages, since they are automatically downloaded and installed.

This packages are downloaded from the Octave packages repository and installed.

Make sure all packages are loaded at Octave startup. One way to achieve auto loading is by editing the ".octaverc" file that is in your home directory.

Include the following lines:

```
pkg load signal  
pkg load communications  
pkg load aria_uwb_toolbox
```

4.1.5 QWT Plot

Compile QWT : e.g. inside the QWT source folder

```
>/usr/lib64/qt6/bin/qmake qwt.pro  
> make  
> make install
```

Make sure the QWT Plot is built using the same Qt Version as the ARIA RDK Qt Version.

Modify the CmakeLists.txt by setting the appropriate include folder for QWT

```
find_library( QWT NAMES qwt qwt6 PATHS /usr/local/qwt-6.3.0/lib REQUIRED )  
set(ENV{QT_PLUGIN_PATH} "${QWT_ROOT}/plugins:$QT_PLUGIN_PATH")  
find_package(VTK REQUIRED)
```

and

```
include_directories(/usr/local/qwt-6.3.0/include)  
install(TARGETS ARIA_RDK
```

4.1.6 Qscintilla

Qscintilla provides lexer capabilities to the Octave script editor.

It can be download and installed according to the Qscintilla building instructions.

Update the following lines in the CmakeLists.txt file.

```
# QScintilla  
target_link_libraries(ARIA_RDK PRIVATE "~/projects/QScintilla_src-2.14.1/designer/libqscintillaplugin.so")  
target_link_libraries(ARIA_RDK PRIVATE "~/projects/QScintilla_src-2.14.1/src/libqscintilla2_qt6.so")
```

4.2 Additional Notes

ARIA RDK require the Octave library and headers to be known and linked during build process.

The actual position of the Octave library and header files may change, depending on the installation path of Octave.

Before compiling, in the CmakeLists.txt the following lines must be updated with actual position of the library files (.dll in windows environment).

```
# Octave 9.2.0
target_link_libraries(ARIA_RDK PRIVATE "/usr/lib64/octave/9.2.0/liboctave.so")
target_link_libraries(ARIA_RDK PRIVATE "/usr/lib64/octave/9.2.0/liboctinterp.so")
```

While the header files folder must be update as well.

```
# Octave 9.2.0
include_directories(AFTER "/usr/include/octave-9.2.0/octave")
```

5 ARIA-RDK Main Concepts

5.1 Workspace

ARIA RDK introduces the concept of workspace, i.e. a folder structure where all the building blocks of a customer application project are defined.

A workspace contains the following elements:

- A set of radar modules definitions: a radar module is a description of one specific UWB module, with the firmware it runs, and its interface with the PC (Fig. 2);
- A set of elaboration scripts: a script is a set of instructions to run customer applications and algorithms. Currently ARIA-RDK support the Octave scientific-programming language;
- A set of radar devices: a device is a physical board attached to the PC. Each device must have its corresponding module definition properly loaded.
- One or more schedulers (handlers): one scheduler is responsible to synchronize more devices to run at the same time, to enable multi-radar acquisition.

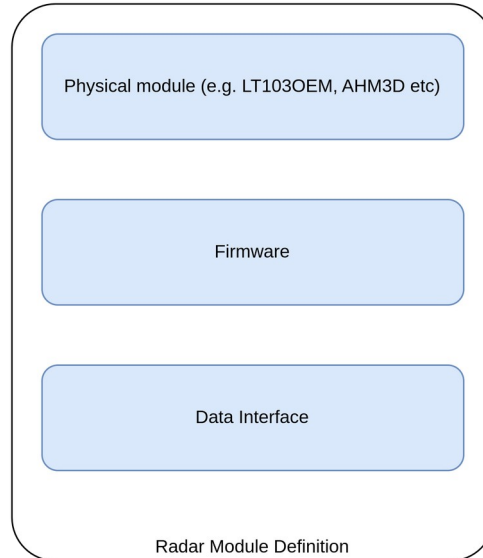


Fig. 2: Radar Module Definition

These components are described in the following sections.

5.2 Radar Modules

One "radar module" is the definition of a peculiar family of radar devices. Each module is uniquely identified by the following data:

1. Its name;
2. A set of parameters which are the radar variables exposed to the PC. They vary according to the radar module and the installed FW. For instance, a module with "basic" FW would provide only the radar's raw-data while the same module with a more advanced FW (e.g. to detect and locate people) would provide the results of the elaboration (e.g. the presence detection outcome and the position of the detected person); see chapter 5.4 for further details.
3. Optional: the definition of the antenna array, with the antennas models and their position to evaluate the beamforming capability;
4. The scripts that are to be executed at start-up, before and after each data acquisition;
5. The configuration of the serial port.

5.3 Radar Devices

A radar device is the physical device connected to the PC. It is the logic representation of a physical device (Fig. 3).

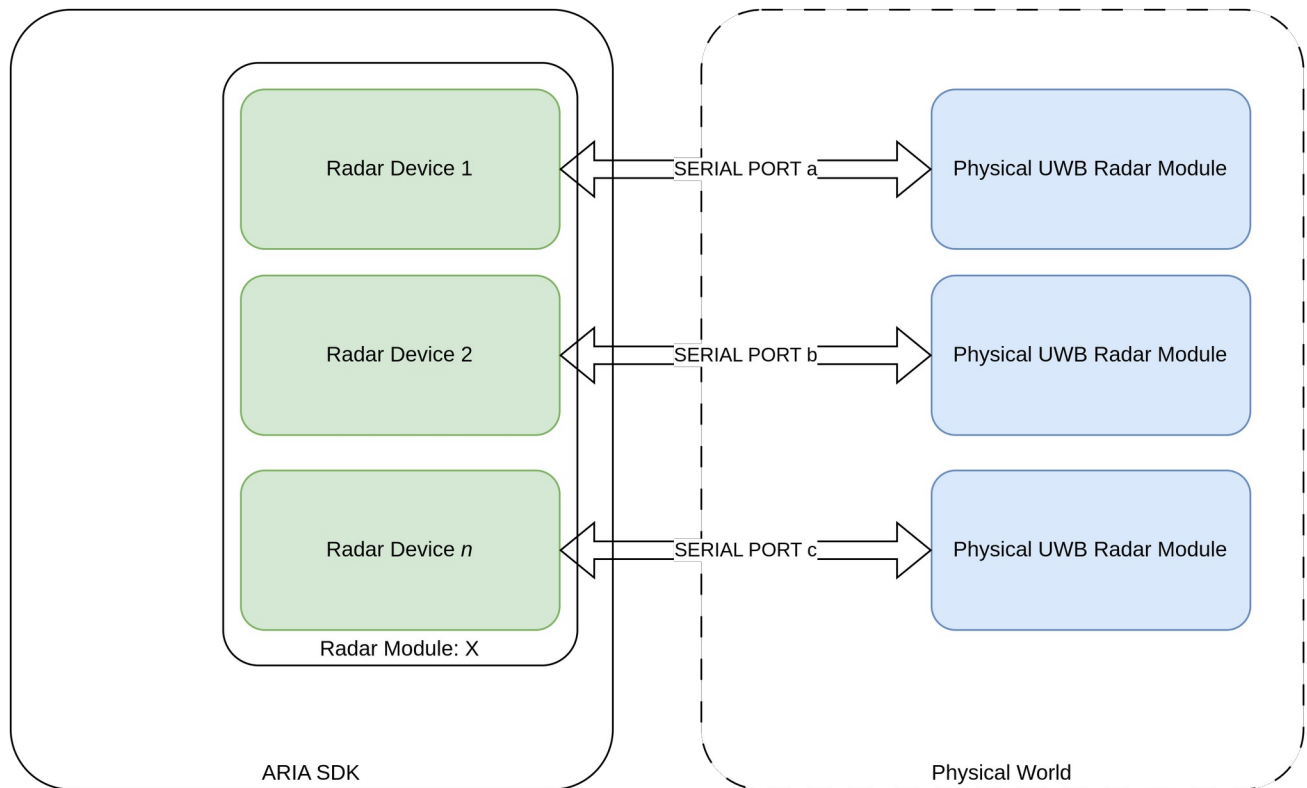


Fig. 3: Logical and Physical Representation of Radar Devices

A radar device is defined by its radar module definition (from which the parameters are inherited) and a set of scripts that are executed

- at start-up (i.e. before the 1st acquisition cycle)
- before any acquisition cycle
- after any acquisition cycle

See below for details on radar acquisition cycles.

5.3.1 Radar Cycle

A radar cycle is made of 4 steps:

emPulse® is an *ARIA Sensing* trademark.

1. init phase: all initialization scripts and variables are set
2. pre-acquisition: all parameters that affect next acquisition are set
3. post-acquisition: this is where customer post-acquisition algorithms are expected
4. frame-rate synchronization

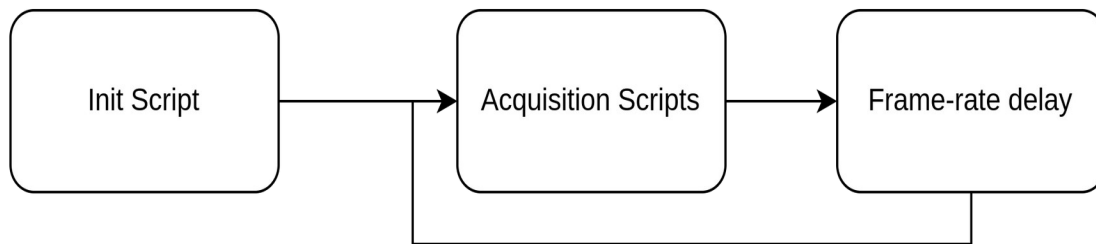


Fig. 4: Basic Radar Cycle

In Fig. 4, the basic radar cycle is shown: after a setup phase ("init"), each acquisition cycle is made of a pre-acquisition phase where the selected scripts are executed and, then, the corresponding parameters are fed into the physical device. When the radar device is able to provide data to the PC and the transfer of the selected parameters is complete, another optional set of scripts is executed (post-acquisition). Please note that the proper acquisition is made by the firmware running into the physical device, so from the RDK perspective we have just two different "moments": before starting an acquisition and after the acquisition has been completed.

If this process (pre-acquisition, acquisition, post-acquisition) is faster than the required refresh rate, a delay is added to maintain the required refresh-rate.

If, instead, the process is longer than the refresh-rate, the behavior of the system is defined into the *scheduler* options. See *Schedulers* for further details.

5.3.2 Policy during init and acquisition cycles

All phases are further split in 2 different steps:

1. Inquiry of output and I/O params and
2. Execution of the scripts

5.3.3 Radar Flow of Operations

Parameter Update

When a parameter is updated, the new value is sent through the serial port to the physical device. It may happen that the actual value stored inside the physical radar differs slightly from

5.4 Parameters

A parameter is a variable that is exposed from the physical device / firmware. ARIA RDK can therefore inquiry for actual value(s) of a parameter, or set its desired value(s).

Basically, each device parameter in ARIA-RDK corresponds to an internal variable in the corresponding radar boards.

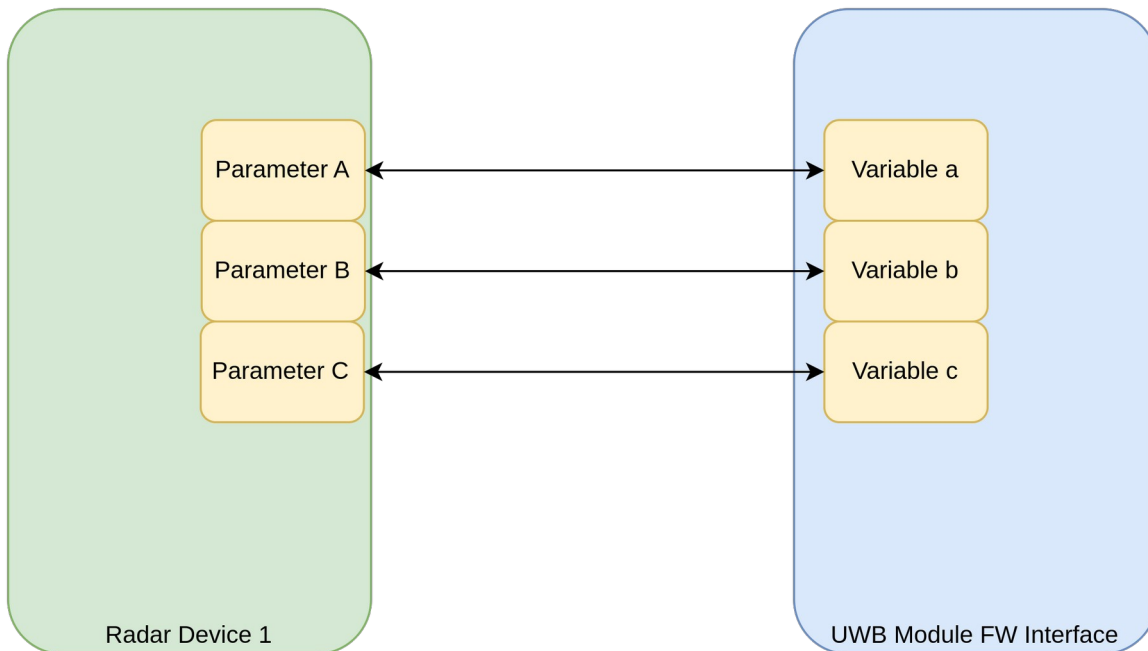


Fig. 5: Parameters and link to physical module's FW variables

5.4.1 Name

Each parameter is defined by a unique, non empty name.

5.4.2 Value

The value of the parameter is the copy of the value of the corresponding variable in the FW of a radar device; clearly, a parameter (and its value) is meaningful only if it is exposed from the radar firmware. Different variables may have different value types. Consequently, the associated parameter defined in ARIA-SDK must contain this information to guarantee proper data transfer/interpretation.

Parameters' values may be of different types, these types are enlisted as follows:

	Description	Data size (bytes)
void	Parameter without data. This kind of parameters are, in fact, pure commands	0
int8,uint8	8-bit integer (unsigned) value or vector	1 * number of elements
int16,uint16	16-bit integer (unsigned) value or vector	2 * number of elements
int32,uint32	32-bit integer (unsigned) value or vector	4 * number of elements
float	32-bit float value or vector	4 * number of elements
char	String or single char	1 * number of elements
enum	Set of available uint8 values with a string descriptor	1 * number of elements

Any parameter (except the "void" case) can be a single value or an array. In case of an *enum* parameter, a parameter value belongs to a set of valid values. Each valid value is described by a unique string. Each element of the valid set is a pair <int8_t, string> where the first part (int8_t field) is the data effectively passed to/from the physical device. The second part (string field) is just a descriptor of the corresponding value.

As an example, the TX POWER of the LT102V2 can have one of the following values:

Value	Descriptor
0x00	TX POWER OFF
0x01	TX LOW POWER
0x02	TX MID POWER
0x03	TX HIGH POWER

ARIA UWB Radar Devices share a common protocol where the radar physical device returns the actual value of a parameter as part of the "command acknowledge" return packet. Basically, when setting the value of a parameter, the radar physical device returns the same command with the value of the corresponding variable after the modification has taken place.

This has a dual objective:

1. It may be used as a confirmation message that the transfer has been successfully interpreted by the radar device
2. Some variables are modified in the radar device according to the desired value but the final value may be different from the desired one for different reasons: for instance, when setting the desired maximum range the LT102V2 rounds the actual value to the closest round-trip distance, sampled according to the sampling speed. With this return value one can check how a device responds to a variable modification request.

5.4.3 Transfer Direction

A parameter may be defined as:

- Input parameter: the value is set only by the ARIA-SDK.
- Input/Output parameters: values can be defined either by the SDK or by the radar device
- Output parameter: values can be defined only by the physical device. The straightforward example is the current radar raw-data. These parameters do not have a value to be attached to the command sent from the ARIA-RDK to the device. Moreover, they don't have an inquiry value.

5.4.4 Valid values

Each parameter (aside from VOID, which has no value associated, and from *enum* type) value can optionally be limited by:

1. a valid min-max interval (a value is valid if and only if it is greater than or equal to the min value and lower than or equal to the max value)
2. a set of valid values (actual value is valid if and only if it belong to said set)

5.4.5 Command String

This is the string that is sent to the physical device through the serial port for parameters inquiry / update.

5.4.6 Inquiry Value

Ideally, when a variable's value is modified by the FW running in a physical module, the corresponding parameter in ARIA-SDK should reflect the actual value.

To reduce the number of transfers over the serial port, ARIA-SDK is responsible for asking the actual value of a parameter, when needed.

To interrogate the actual value of a variable, the corresponding parameter is provided with a "special value". This peculiar value is called the "Inquiry Value". When sending the "inquiry-value" to the radar physical device, the latter does not change the variable corresponding to the parameter. Instead, it returns the actual value of the internal variable.

5.4.7 Command String

This is the string sent along with the payload (either the set-value or the inquiry-value) to allow the radar physical device to identify which parameter/variable is involved in the transfer.

5.4.8 Grouped Values

Some parameters are grouped in several sub-parameters and transferred at once at every single command.

For instance the radar version for LT102V2 is defined by 3 fields which are handled as 3 different parameters

- Version (reserved) [uint16]
- Version_hw_id [uint16] which is a word describing the HW board
- Version_fw_id [uint32] which is a word describing the FW identifier

6 Data Workspace

The data workspace is a set of variables that can be read from the radar devices, or modified / create during scripts execution.

It also provides a seamless link between a radar device, and its parameters and the scripts that are executed by the RDK itself.

7 Scripts

A script is a set of instructions to be performed on radar or application data. The underlying elaboration tool and interface is *Octave* (<https://octave.org/>)

ARIA-RDK provide an active link to/from the *Octave* interpreter; this allows for great flexibility in algorithms description / prototyping.

8 Antennas

The ARIA-RDK allows to handle antenna radiations properties. In UWB systems, antennas may be as important as the front-end: differences in group delay over azimuth and elevation should be taken into account when evaluating complete UWB systems. For this reason, ARIA-RDK is able to read FFIO files exported from EM-simulation tools.

FFIO files contains information about electric field, in the *Theta, Phi* polar representation, radiated from an antenna, at a reference distance of 1m to the antenna.

9 Getting Started

Once ARIA-RDK is properly installed, it starts with an empty environment (Fig. 6). The on the right part of the main windows, the Octave Interface window is shown.

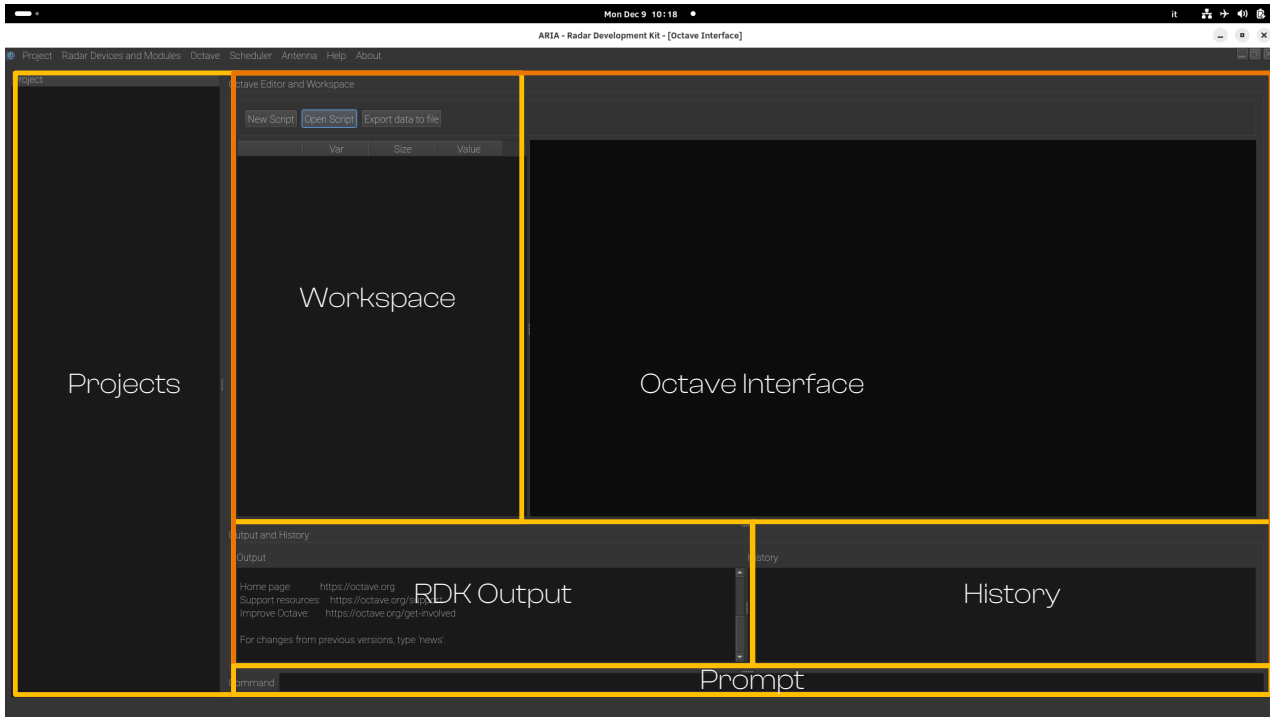


Fig. 6: RDK Appearance at Start-up

9.1 Octave Interface

The "octave interface" window is the window to write/load/save and execute scripts, plot variables, execute prompt-commands.

NOTE: as of release 0.1, no "debug" function is available during scripts execution.

With a link to the versatile Octave engine, a user can exploit also different packages/toolboxes to perform different tasks, from data processing to advanced applications.

On the right side of the "octave interface", a table (*Workspace* in Fig. 6) containing all variables is shown.

9.2 Workspace

The workspace section contains all the variables that are part of the current project. These variables may be created / updated by an Octave script or can be the

9.3 Scripts

Scripts are ".m" files that can be interpreted / executed by the linked *Octave* library.

9.3.1 Create/open a new script

From the Octave interface window, click respectively New Script or Open Script on the top-left corner of the Octave Window (Fig. 7).

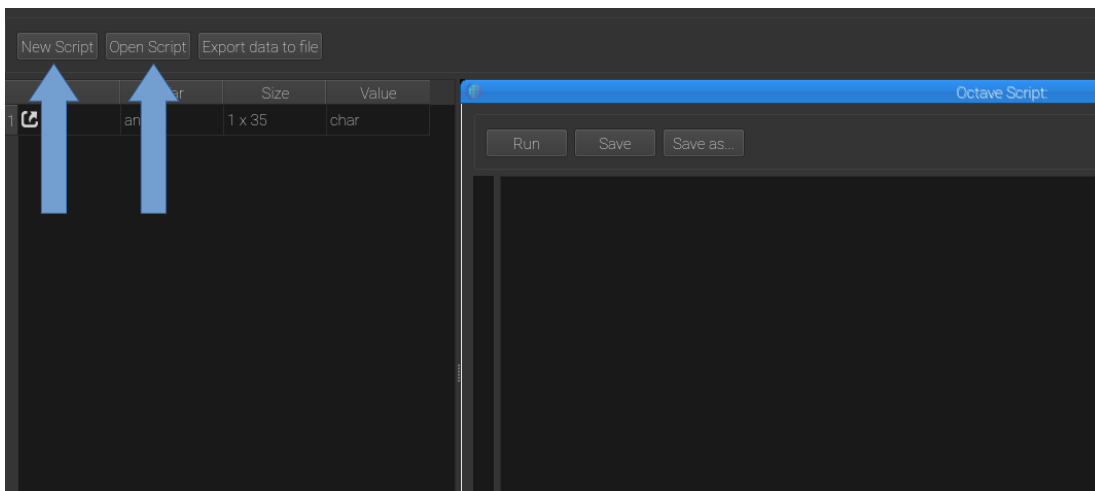


Fig. 7: Create or Open a script

9.3.2 Run a script

Once a script is available, it can be run simply by pressing the "Run" Button (Fig. 8). The workspace is updated immediately after the script execution.

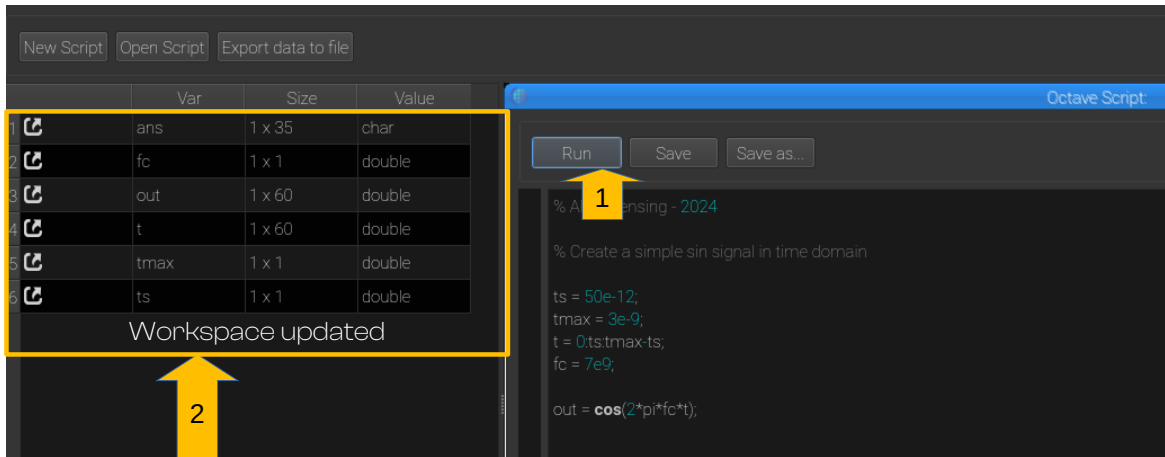


Fig. 8: Script execution

9.4 Prompt commands

In the bottom line of the Octave Interface Window, users can type any octave command to be executed immediately. Once the command is done, the workspace is updated immediately.

NOTE: contrary to normal Octave interface, "executing" a variable name do not display its value in the output window. To print out the content of variable, type

`var_name = var_name[ENTER]`

In the example below: `ts` is a single double value (`ts = 5e-12`), to print it out, in the command prompt `ts = ts` command is issued.

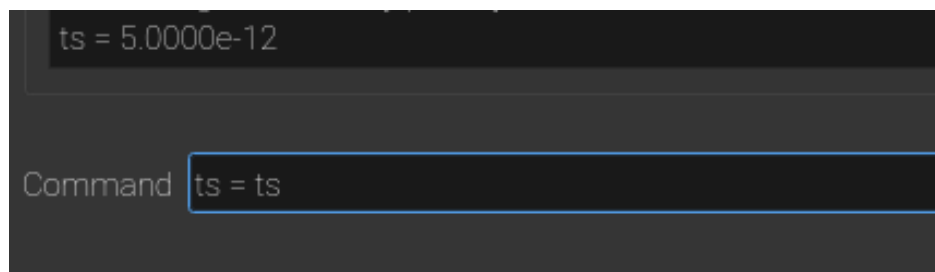


Fig. 9: Command to print-out the variable `ts`

9.5 Plots

The ARIA-RDK provides a set of plots:

emPulse® is an *ARIA Sensing* trademark.

- Line Plot
- Scatter Plot
- Box Plot
- Area Plot
- Bar Plot
- Density Plot
- Arrow Plot
- 2D Vector field (e.g. *quiver*).

9.5.1 Line Plot

Line plot is a plot where a set of (X,Y) data is represented. A segment is added between each consecutive (X,Y) pairs. If no X data is given, X is assumed being an integer from 0 to N-1 where N is the number of samples of Y vector.

9.5.1.1 Create a Line Plot

To create a Line Plot, select one or more variables in the workspace table.

Right-click and a pop-up menu will appear. Select **plot** and one of the following voices:

- new plot (each curve into own plot)
- new plot (all data into the same plot)
- new plot (x,y1,...,yn)

When "new plot (each curve into own plot)" is clicked upon, each variable is associated with Y values of the plot, and X data is created as vector of values in the [0,N-1] range, where N is the number of samples of the corresponding Y vector. Each set of data is plotted into a single dedicated plot.

For example, if the following script is executed:

```
% Create a simple sin signal in time domain  
  
ts = 5e-12;  
tmax = 1e-9;  
t = 0:ts:tmax-ts;  
fc = 7e9;
```

```
out = sin(2*pi*fc*t);  
o2 = 5*cos(3*pi*fc*t);
```

When selecting "o2" and "out" from the workspace table, the following plot is produced (Fig. 10).

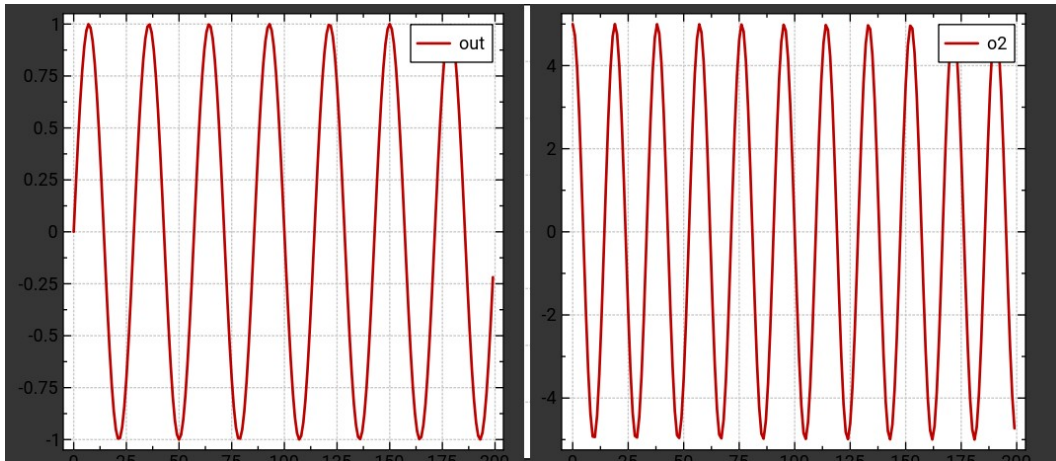


Fig. 10: new plot (each curve into own plot)

When "new plot (all data into the same plot)" is clicked upon, each variable is associated with Y values of the plot, and X data is created as vector of values in the $[0, N-1]$ range, where N is the number of samples of the corresponding Y vector. Plotting the example above:

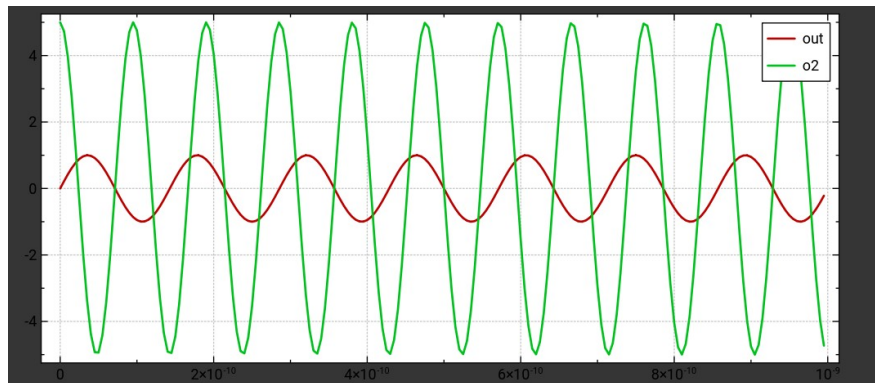


Fig. 11: new plot (all data into the same plot)

When selecting **new plot (xy1, ..., yn)** the first variable selected is associated with X-values, each other vector is associated with Y values. In this case all vector must have the same number of values.

Note: X data does not to be monotonic.

emPulse® is an *ARIA Sensing* trademark.

(e.g. when selecting *o2* and then *out*, the following plot is created - Fig. 12).

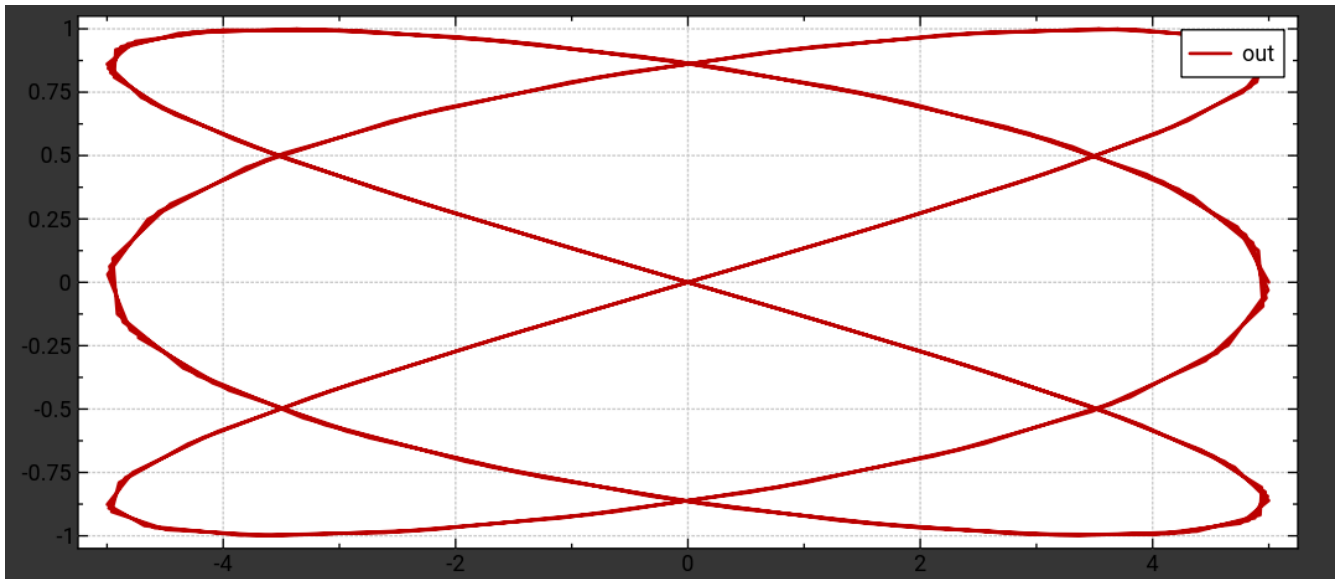


Fig. 12: new plot (x, y_1, \dots, y_n)

If one of the selected variables is a matrix, then a curve is created per each column or row of the matrix. The current selection (rows / columns major) is done according to the following policies:

1. if no X data is given, the selection is done in order to minimize the number of curves: e.g. if a matrix is sized 200 rows, 2 columns, one plot is created per each column. If a matrix is sized 2 rows, 200 columns, one plot is created per each row.

If, instead, X data is given, one of the two dimensions of the matrix must coincide with the length of X data.

9.5.2 Scatter plot

Scatter plot is identical to the Line Plot (see 9.5.1) except for the fact that points are not connected by a line segment but a symbol is drawn over each (X,Y) pair.

The above example is re-plotted in the next pictures

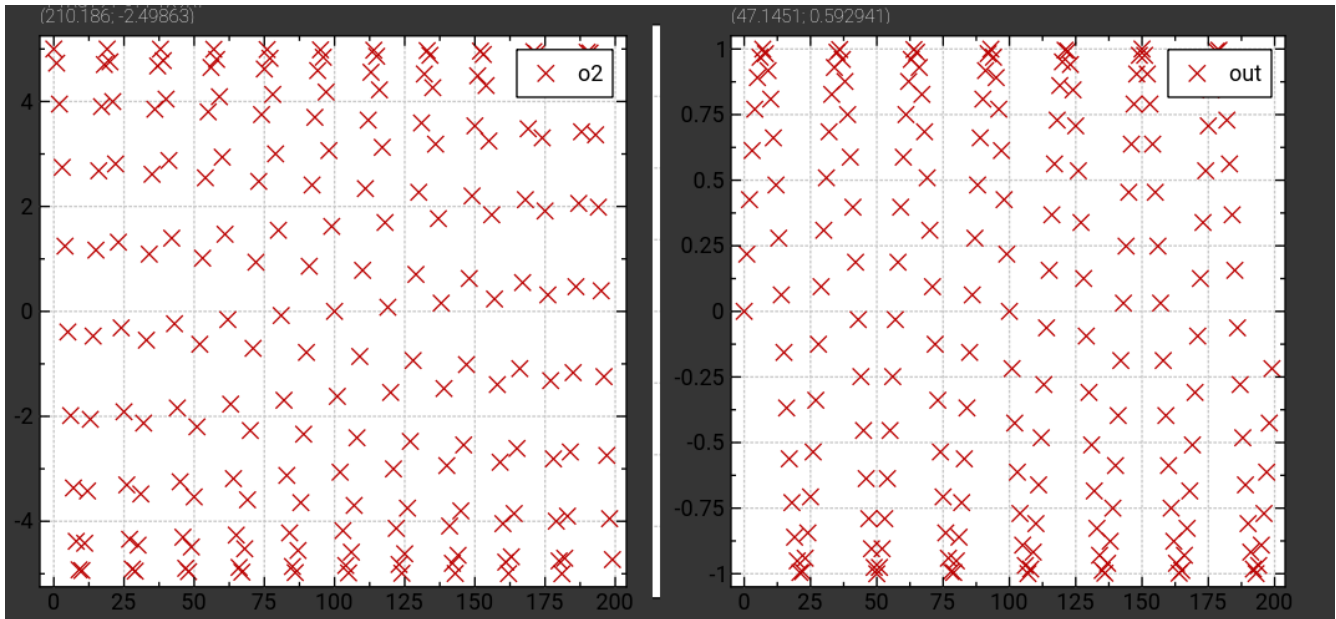


Fig. 13: new scatter plot (each curve into own plot)

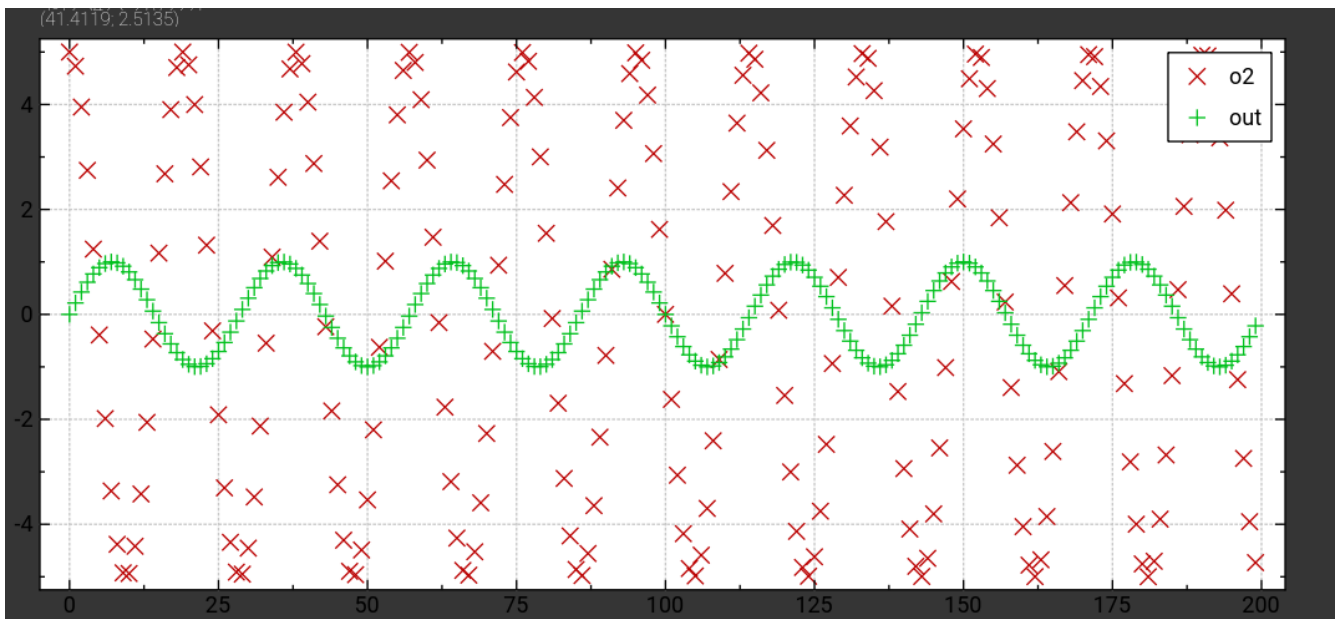


Fig. 14: new scatter plot (all data into the same plot)

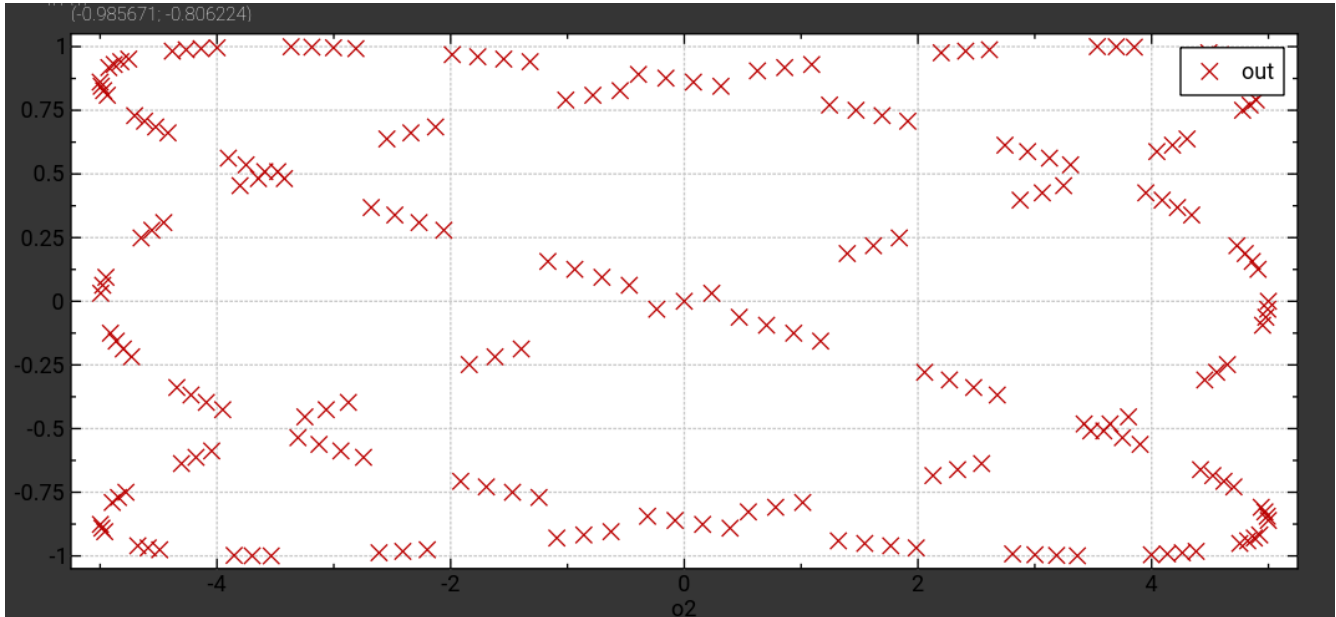


Fig. 15: new scatter plot (x, y_1, \dots, y_n)

9.5.3 Box-plot

Given a vector, the box-plot shows the statistical summary of the data contained (Fig. 16):

1. Mean value: it is represented by a small cross
2. Median: it is represented by the continuous line in between the main box
3. First Quartile (Q25): it is the lowest boundary of the main box
4. Third Quartile (Q75): it is the upper boundary of the main box
5. Max Value: it is the highest value in the input data lower than $Q75 + 1.5 * IQR$
6. Min Value: it is the lowest value in the input data higher than $Q25 - 1.5 * IQR$
(IQR is the Inter-Quartile Range: $Q75 - Q25$)
7. Outliers: all values greater than $Q75 + 1.5 * IQR$ or lower than $Q25 - 1.5 * IQR$

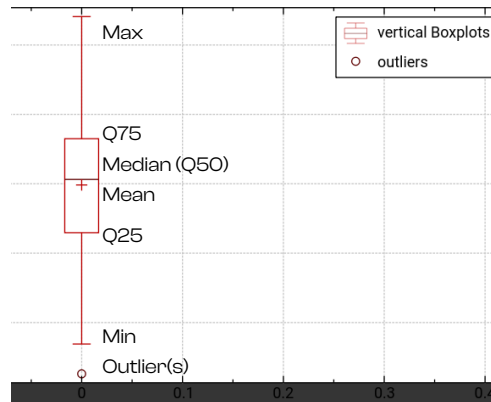


Fig. 16: Box-plot description

9.5.4 Area Plot

Area plot is similar to Scatter and Line Plots. A filled area is drawn from line $Y=0$ to each Y value.

The previous example is plot as follows

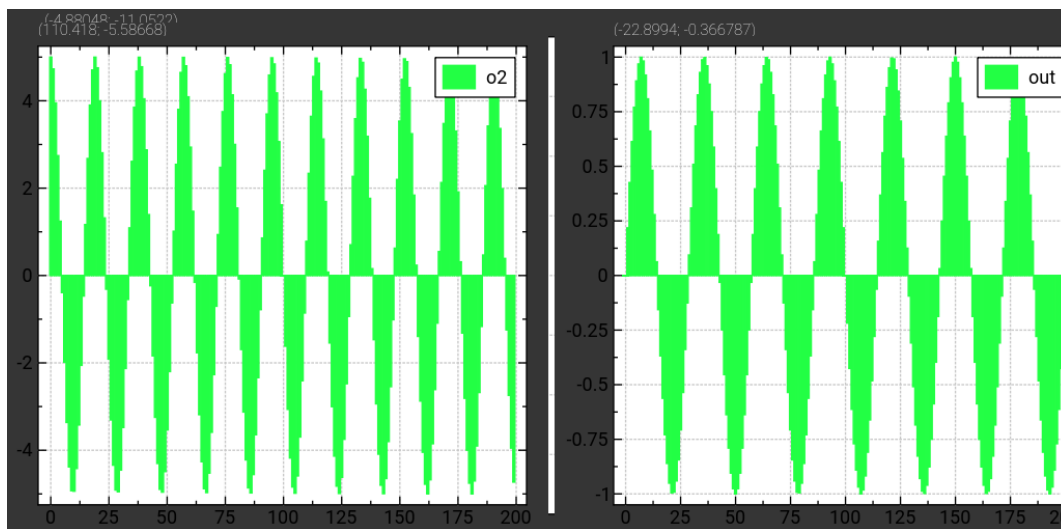


Fig. 17: Area Plot

9.5.5 Arrow plot

Similar to the above plots, an arrow is drawn from the $(X,Y=0)$ reference axis to the (X,Y) point. The above example is shown in Fig. 18.

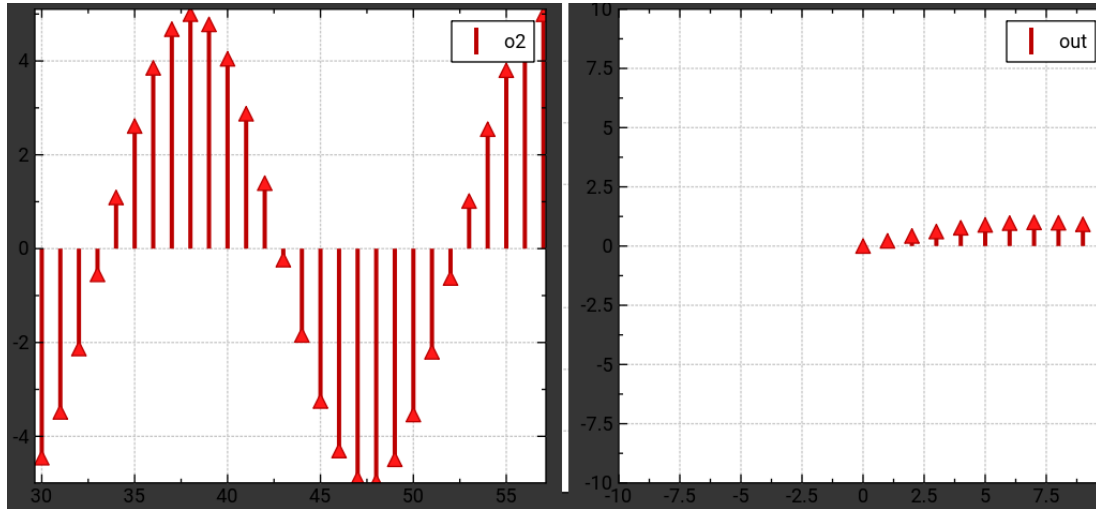


Fig. 18: new arrow plot (each curve into own plot)

9.5.6 Density Plot

This is a 2D plot where a matrix is represented with a 2D density plot, adjusted to the data range.

In the example below the output Density plot is shown,

```
x = 0:1e-2:1;
y = 0:1e-2:1;
[xx,yy] = meshgrid(x,y);
z = cos(xx * 2* pi) + sin(yy*6*pi);
```

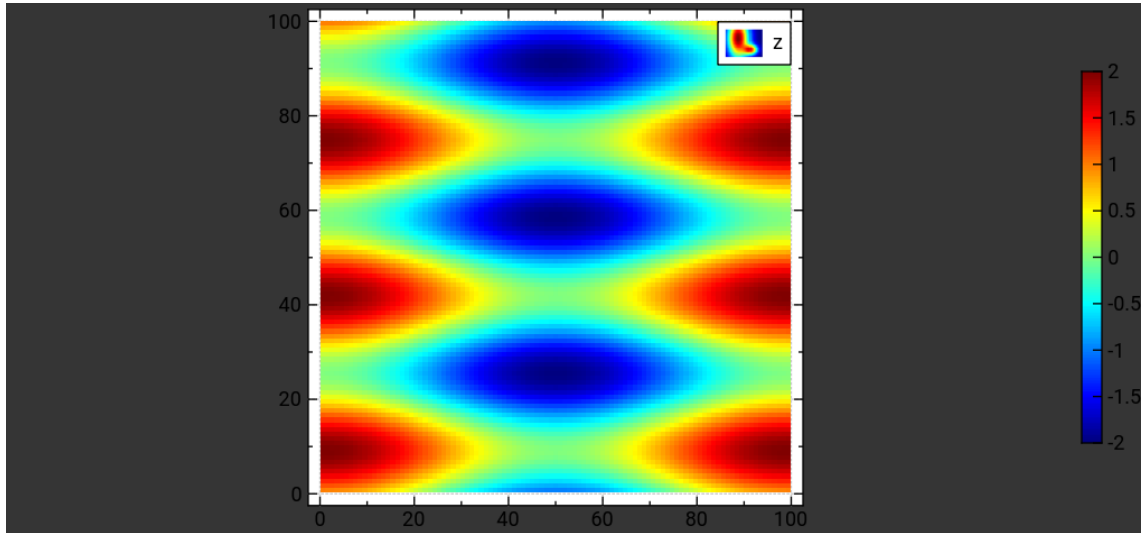


Fig. 19: Density plot example

9.5.7 2D Vector Field

The vector field (also known as *quiver*) is the representation of a 2D vector field. Input data is made of two matrices (they must obviously be the same size). The first one is the x- component of the field, the second one is the y- component of the field.

As an example, the following code

```
x = 0:3e-2:1;
y = 0:3e-2:1;
[xx,yy] = meshgrid(x,y);
e_x = cos(xx * 2*pi);
e_y = sin(yy*6*pi);
```

produces the following plot (e_x, e_y)

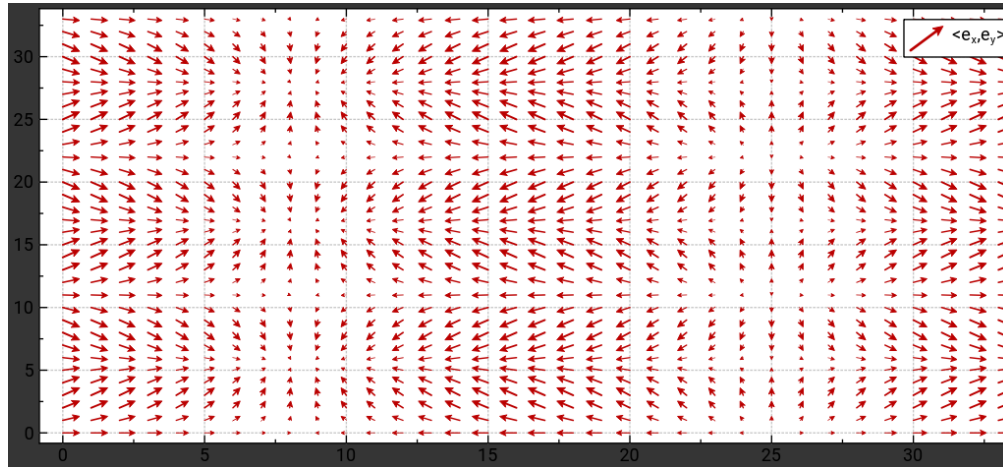


Fig. 20: 2D Vector Field

9.5.8 Data Update

Every time a variable is updated, any plot referring to the variable is updated in real-time.

9.6 Projects

9.6.1 Project structure

A project is contained in a folder and it's based on a defined folder tree (Fig. 21)



Fig. 21: Project Structure

The project folder needs one file *arp* (ARIA Radar Project) which is an XML file containing all the information of the different elements belonging to the project. This is the project main file.

The project's folders are:

- **Antenna Models:** in this folder there are all antenna model files owned by project's radar modules
- **Handler:** in this folder there are all device scheduler files
- **Protocols:** for future releases
- **Radar Devices:** contains all the device definitions. These are the devices connected to the RDK and used by the projects
- **Radar Modules:** contains all the radar modules description files.
- **Scripts:** contains all the scripts used by the projects

9.6.2 Create a new project

Click on Main window's menu "Project", "New".

A file dialog appears, asking for the project main-file name.

The extension "arp" is added to the file, if not given or a different extension is provided.

NOTE: a folder with the main-file name (without any extension) is created and the main file is created in said folder.

In the same folder, the sub-folder structure is initialized.

9.6.3 Open an existing project

Click on Main window's menu "Project", "Open" and select a valid project main file.

9.6.4 Save a project

Click on Main window's menu "Project", "Save".

9.6.5 Close a project

Click on Main window's menu "Project", "Close".

9.6.6 Copying / Importing a new project

A project can be easily exported simply by copying the main files and all the project folders (along with their content) into the desired destination.

9.7 Radar Modules

Radar Modules definitions are stored in the "Radar Modules" folder of the project. Each module definition is stored in a file with extension "arm" (ARIA Radar Module) which is an XML description of the module properties.

9.7.1 Create a new radar module

Once a project is open, it is possible to create radar modules definitions from scratch.

Click on main window menu "Radar Devices and Modules", "Radar Modules", "New Radar Module".

The radar module editing window will appear (see paragraph 9.7.4).

9.7.2 Import an existing radar module

Once a project is open, it is possible to import radar modules definitions.

Click on main window menu "Radar Devices and Modules", "Radar Modules", "Import Radar Module".

An open-file-dialog will pop-up; select a valid radar module file.

NOTE: when importing an existing radar module, if its scrips or its antennas are not present into the project, they will be reset.

9.7.3 Edit a radar module

To edit a radar module from the project tree, double click on its name. The radar module editing window will appear (see paragraph 9.7.4).

9.7.4 Radar Modules Editing Window

The radar modules editing window is shown in Fig. 22.

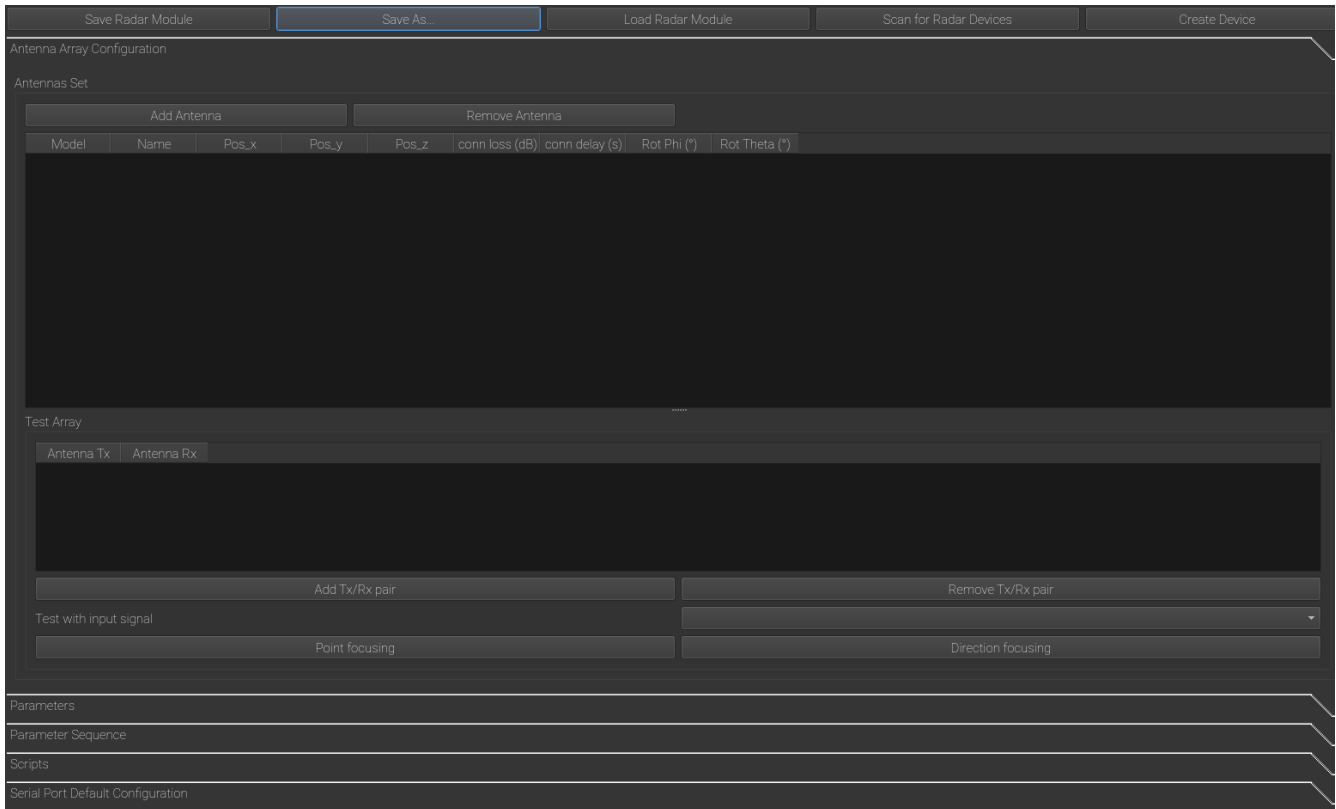


Fig. 22: Radar Modules editing window

This window is made of 5 different tabs, to edit the different properties of a radar module:

- antennas set
- parameters
- parameter sequence
- scripts
- serial port default configuration

9.7.4.1 Defining antennas

In the "Antennas Set" tab it is possible to define the array of actual antennas, with their position on the board module. This allows for further evaluation of radar properties and for preliminary evaluation of the performance of a radar module, in terms of focusing capabilities.

To define an array add as many antennas as needed.

Each antenna of the array may be associated to an antenna dataset (Paragraph 8); in case the antenna model is not given, no further assumption is made and the antenna is internally marked as "no data".

Each antenna must be given a unique name and its position. This is needed to calculate "direction focusing" properties; i.e. the radar beam obtained combining all the antennas with relative phase-shift calculated to focus the beam over a desired azimuth / elevation angle.

It is not mandatory to define antennas but this function may be useful for prior evaluation of custom arrays.

When a new antenna is added, it is possible to declare its reference radiation dataset, by clicking on the first field of the newly created row (Fig. 23).

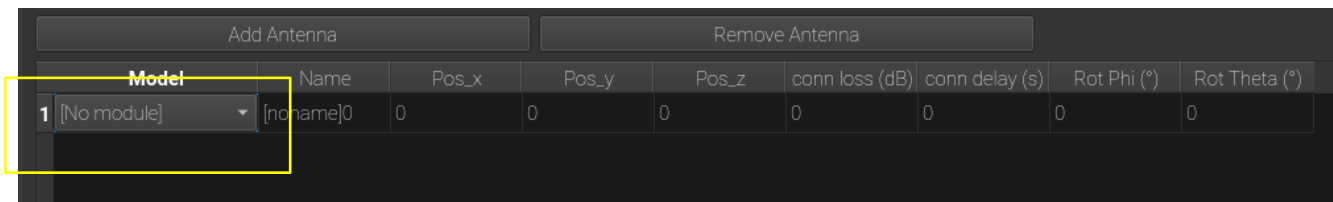


Fig. 23: Antenna model selection

When clicked, the list of available antennas is shown. It is possible to load a new one, or leave "no module". When "no module" (default option) no information is available for the current antenna.

If a set of antennas is defined, it is possible to declare the combinations that are excited during beam-forming; these are expressed as a set of Tx, Rx antennas pairs.

In each pair, Tx and Rx antennas may also be the same antenna (monostatic) or different (bistatic).

NOTE: before editing the array of Tx/Rx antennas combinations, the antenna arrays must be defined entirely; save the radar module before editing the array.

When a set of Tx/Rx pairs is defined, it is possible to evaluate its array properties, by clicking on "Direction Focusing". The underlying model is the Phased-Array: different antennas are fed with properly phase-shifted tones, so that the beam is focused onto a desired azimuth, elevation angle.

For UWB-Time Domain calculations, refer to the [ARIA_uwb-toolbox](#) documentation (Paragraph 12.2).

9.7.4.2 Definining parameters

To define the different parameters of a radar module, open the "Parameters" tag (Fig. 24).

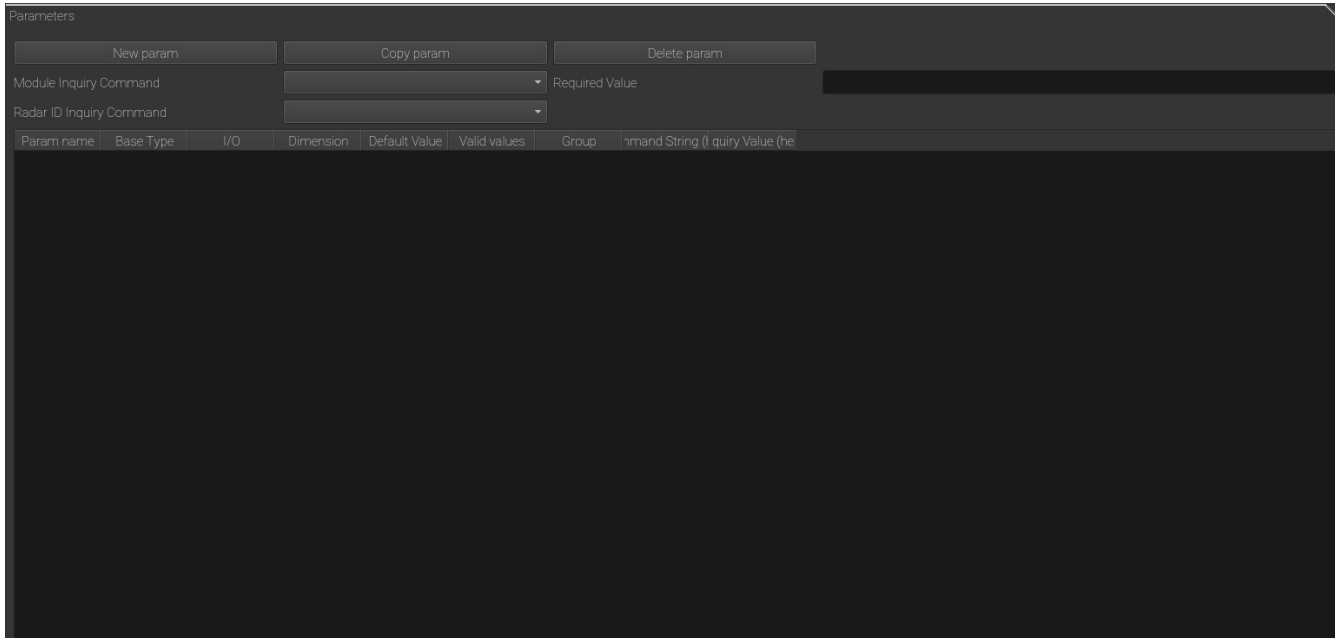


Fig. 24: Module parameters editor

To add a parameter, click on "New Param" button. A new line in the param table is created (

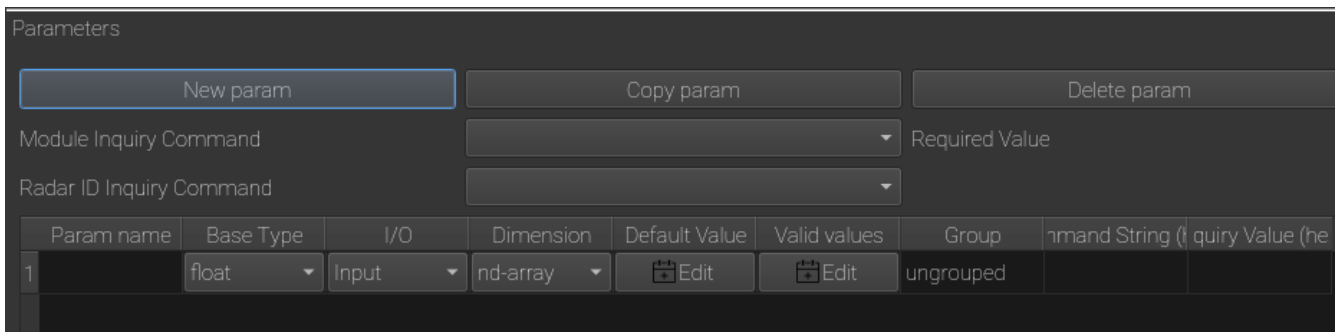


Fig. 25: Param line editor

Fill the line with all parameter's properties: name, data type, direction, group (optional), command string and (optional) inquiry value.

9.7.4.3 Valid Values

If a parameter value may belong to a limited set of values, it is possible to define its valid range by clicking on the "valid values" button in the parameter line.

The valid values may be defined in two ways:

emPulse® is an *ARIA Sensing* trademark.

1. a list of all valid values
2. define min/max range

When the "valid values" button is clicked, the following window appears (Fig. 26).

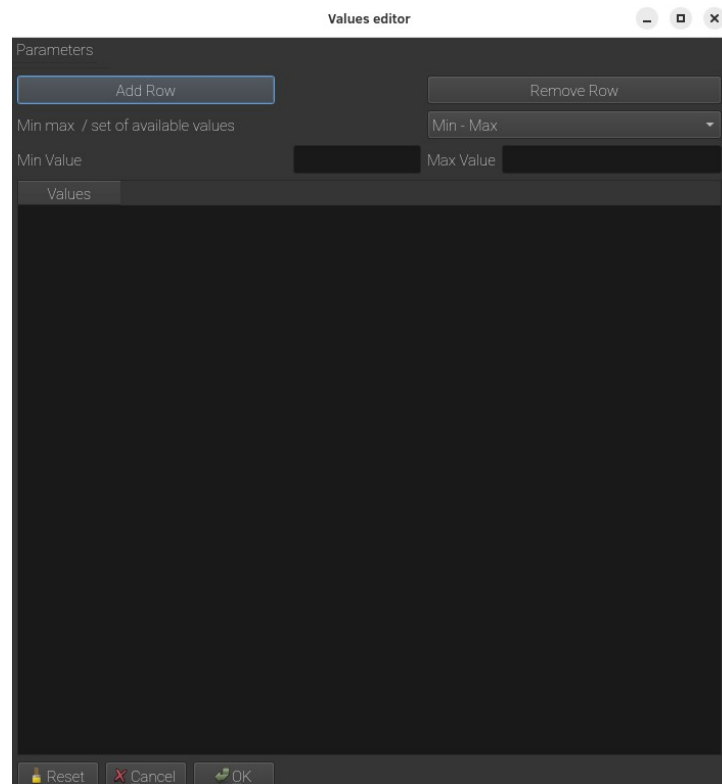


Fig. 26: Editing valid values for a parameter

To define all valid values, click on the drop-down box "min-max" and set it to "list of available values" (Fig. 27)

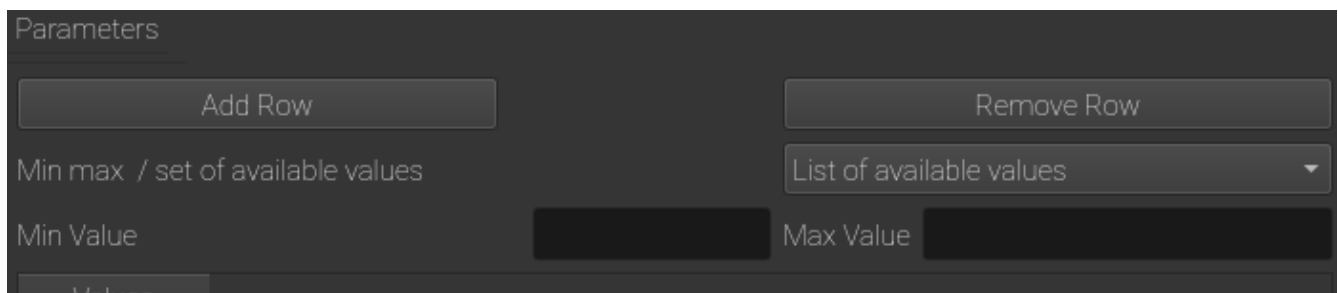


Fig. 27: Edit valid values as a list

Click "add row" for each valid value and fill its rows.

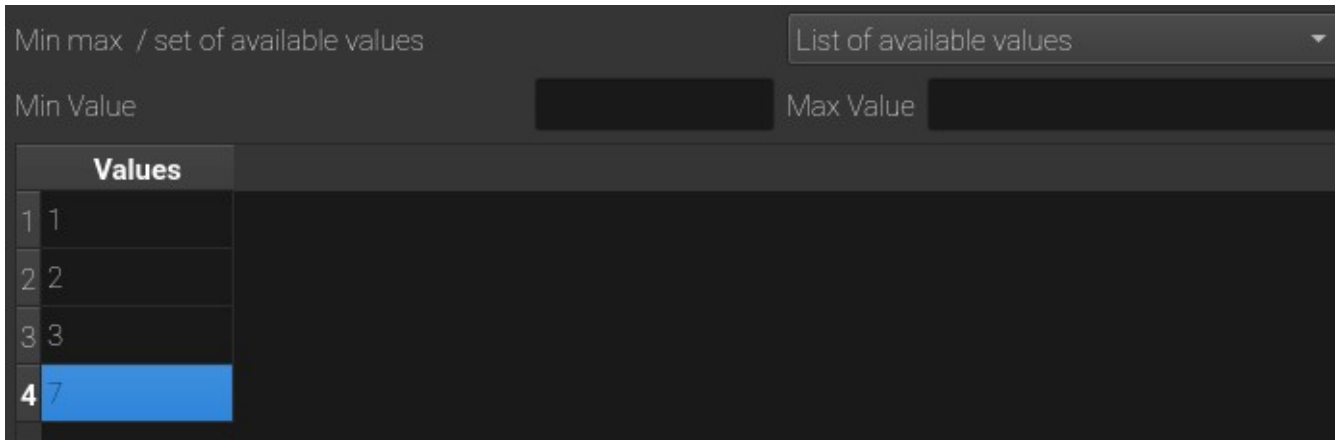


Fig. 28: List of valid values

As an example, in Fig. 28 a parameter may assume a value in the set [1 2 3 7].

If, instead, to define the valid range, select "min-max" in the drop-down box.

When doing so, you may edit min value and max value in the corresponding editing boxes.



Fig. 29: Editing range

As an example, in Fig. 29 a valid range [0..7] is defined.

9.7.4.4 Grouped parameters

It may happen than more than one parameter are grouped and transferred at once during communication.

This is typical when more parameters are related to the same physical measurements and each parameter may be seen as a data field for that reading. For instance, the "version" parameter is made of 4 different fields: a "reserved" field, the hardware code and the firmware id.

Offset	Type	Description
0	Uint16	Reserved

2	Uint16	Hardware code (identify the HW type)
4	Uint8	FW version 1
5	Uint16	FW Version 2

To specify the parameters to be grouped in a single transmission, and to define the order of the parameters to build the transfer payload, the command string must be the same for all parameters of the group and the position of each parameter in the payload must be indicated in between square brackets.

For example (Fig. 30), the four parameters are grouped by the common command string (*0x01*), and they are order by postponing *[position]* to said command string.

As a result, the inquiry of the "version" group will result in the following order:

1. **version_cid**, one uint16 value
2. **version_hwcode**, one uint16 value
3. **version_id0**, one uint8 value
4. **version_id1**, one unit16 value

16	version_cid	uint16	Output	scalar		Edit	ungrouped	01[0]
17	version_hwcode	uint16	Output	scalar		Edit	ungrouped	01[1]
18	version_id0	uint8	Output	scalar		Edit	ungrouped	01[2]
19	version_id1	uint16	Output	scalar		Edit	ungrouped	01[3]

Fig. 30: Grouping and ordering parameters

When grouping different parameters into one single group, each of the "sub" parameter can be independently accessed and modified by the *Octave* environment.

9.7.4.5 Variable-length payloads

ARIA-RDK handles payloads whose length may vary across different readings. For instance, the number of samples of radar raw-data depends on the current detection range. If the range is modified in-between two readings, the resulting vectors will have different lengths. When receiving a variable-length payload, the RDK will check that the data length is an integer multiple of the

number of bytes required by the data-type of the parameters. In case of grouped parameters, only the last sub-parameter can be of variable length.

9.7.4.6 Defining scripts

As reported in Fig. 4, at each step of the radar cycle, it is possible to run a set of scripts. These scripts are defined by clicking on the *Scripts* tag of the Radar Module editor (

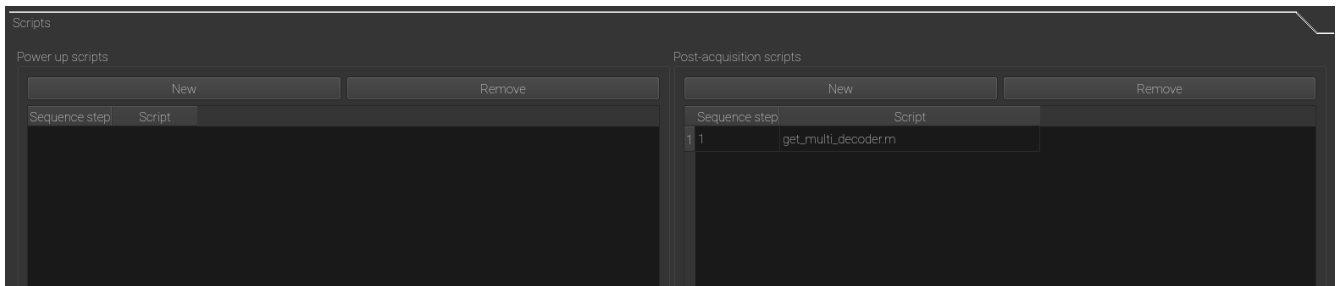


Fig. 31: Setting the scripts to be executed at initialization and at each radar cycle

To add a new script, click on "new" button and a File Open dialog will appear. Select the script file you want to include in the project.

9.7.4.7 Defining serial port

The serial port parameters are defined in the corresponding Tag (Fig. 32)

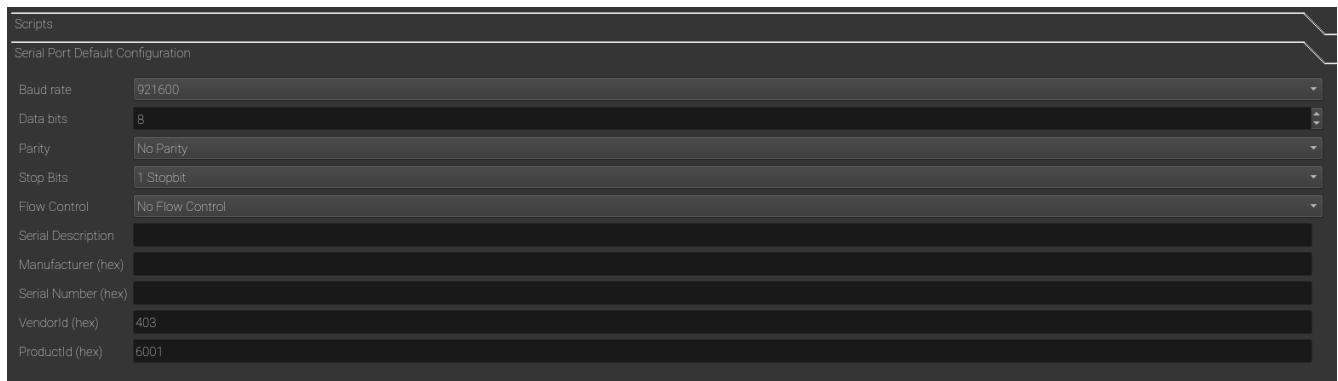


Fig. 32: Serial Port parameters

9.8 Radar Devices

Radar devices are actual instances of defined radar modules.

emPulse® is an *ARIA Sensing* trademark.

9.8.1 Create a new radar device object

To create a new radar device, from the radar module editor you may

1. click on "create device" button
2. scan for connected devices if the device you want to add to the project is already connected to the PC (9.8.2)

In the first case, the radar device editor window appears (see Paragraph 9.9).

9.8.2 Scan for devices

When a device is connected to the PC, it is possible to scan for it, in order to create its radar device object in the current project.

The "scan devices" window is shown in the next picture (Fig. 33).

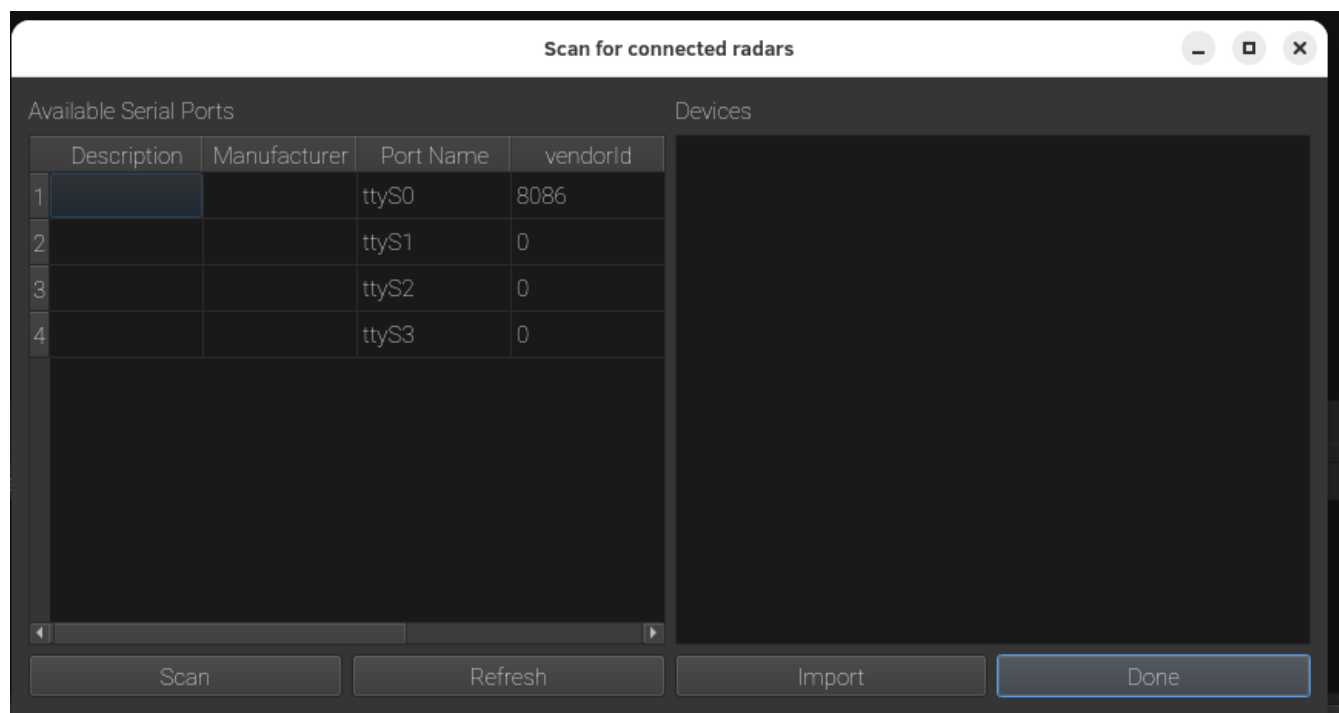


Fig. 33: Scan devices

On the left side all the available serial ports are enumerated. To detect a radar device, select the desired serial port and click "Scan".

If a radar device is attached to the serial port, an entry on the right side is created. To create the corresponding view in the project, select the desired device and click "Import".

9.8.3 Delete a radar device

To delete a radar device, select the device you want to delete from the project and click on the main menu: "Radar Devices and Modules", "Delete"

9.9 Radar Devices Editor

The Radar Device Editor is shown in Fig. 34.

You may open this window either by:

1. Double-clicking on a radar device of the project
2. Selecting "Create Device" from the Radar Module Editor.

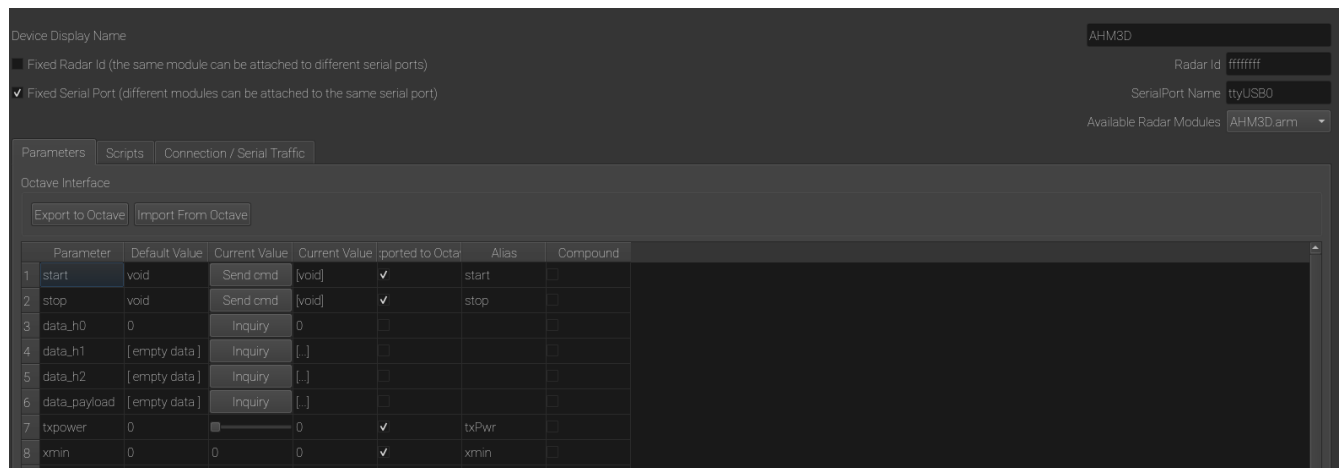


Fig. 34: Radar Device Editor

The Radar Device Editor allows you to define:

1. Which parameters are exported into the octave environment
2. Which scripts are to be executed at radar device startup or during the acquisition cycle

It provides also a convenient serial port monitor for debug purposes.

9.9.1 Radar Device Parameters

In the *Parameters* tag the list of parameters (defined in the corresponding radar module) is shown. Per each parameter, its default value is reported, along with an interactive element to modify (if the parameter is not set as output) its current value.

It also allows to define the name under which the parameter is exported to the Octave environment and if this name must be joined with the radar name.

In the Radar Device Parameters sub-window, the user may

- send commands (a command is a void-payload, input-only parameter) to the radar device,
- inquiry parameters values
- update parameters values by one of the mechanisms provided in the "current value" column
 - Drop-down menu when the parameter is an enum type or has a limited set of valid values (see Paragraph 9.7.4.3).
 - Slider for integer types
 - Line Edit for floating values

When the device is connected (see Paragraph 9.9.4) when a parameter is modified or inquired, its value is immediately transferred to (or inquired from) the radar device.

Please note that, during parameters update, the radar device may return a value that is slightly different from its set point. This may happen, for instance, when a floating value is internally rounded in the physical device itself: the physical device will return the closest value to the desired set point. For instance, the max range is defined as a floating value but, internally, it is rounded to the minimum distance bin. For this reason, the return value may be slightly different.

9.9.2 Compound names

When more than one device of the same definition (radar module) is attached to the same computer, the parameters that are exported could be conflicting. To resolve in a unique way their names, a unique name may be provided to each device's parameter. Or, simply, the "compound name" checkbox may be selected.

For instance, if two devices (e.g. "radar1" and "radar2") are connected and both have the same parameter (e.g. "dataout"), when checking the "compound name", the two different variables will be created: **radar1_dataout** and **radar2_dataout**.

9.9.3 Scripts

In the "scripts" tag, the list of scripts to be executed at init, or during the radar cycles is reported. The lists are pre-populated with the scripts defined in the radar module but the actual execution list can be arbitrarily modified on each device.

9.9.4 Serial Port Monitor

The "Connection / Serial Traffic" is a convenient window to run the radar cycle (see Paragraph 10) and/or debug the serial port traffic. By default, it provides a live feedback of the data transferred thru the serial port (unless the *"Echo Serial Data"* checkbox is left unchecked).

In the serial port monitor tag, a set of buttons are present (Fig. 35).

- *"Scan"*: to scan for devices attached to the computer
- *"Connect"*: to open the connection between the computer and the physical device through the serial port
- *"Init"*: to perform the init steps and then halt the radar cycle
- *"Run"*: to perform the complete cycle.

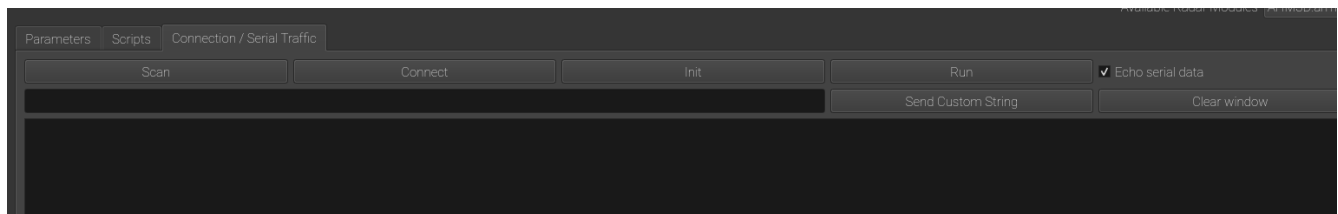


Fig. 35: Connection / Serial Traffic

In this window, the actual data stream to/from the radar device is sent to the text output. It is possible to avoid this stream by unchecking the "Echo Serial Data" check box.

It is also possible (when the connection is successfully established) to send custom byte sequences, for debugging purposes.

9.9.5 Modifying parameters inside a script

To access a parameter (to inquiry its value, to send a command, or to update a device parameter), three special Octave functions (these are found in the **ARIA UWB Toolbox**) are provided. These are:

- `var_immediate_inquiry("varname")`

- This function temporarily halts the execution of the current script and inquiries the *varname* parameter. Note that "*varname*" must be a string representing the alias name of the parameter. If compound name is selected, *varname* must be equal to the full parameter alias.
- **var_immediate_command(*varname*)**
 - This function temporarily halts the execution of the current script and sends the *varname* command. Note that "*varname*" must be a string representing the alias name of the command. If compound name is selected, *varname* must be equal to the full parameter alias.
- **var_immediate_update(*varname*)**
 - This function temporarily halts the execution of the current script and updates the *varname* parameter with the current value of the "*varname*" variable. Note that "*varname*" must be a string representing the alias name of the parameter. If compound name is selected, *varname* must be equal to the full parameter alias.

Example:

```
var_immediate_command("stop");  
var_immediate_inquiry("xmax");  
if (xmax < 3)  
    xmax = 3;  
    var_immediate_update(xmax);  
endif
```

In the above example, we assume that a radar device (e.g. radar1) has an exported parameter (e.g. range max) whose alias is "xmax" and no compound name is specified. The radar device has also a command exported under the name "stop".

In the first row, the "stop" command is sent to the device. Then the current value of the range max stored into the physical radar device is retrieved. In the fourth row, the variable is set to a certain value. This value is then sent to the radar device. The actual value of "xmax" at the output of "var_immediate_update" may be different from the set target (3) due to the fact that the radar device may trim the value to its closest available value (see paragraph 9.9.1).

10 Scheduler

The scheduler is a handler of radar device. Basically the scheduler allows a user to attach different radar modules to the PC and to run them simultaneously. You may add to a scheduler to the project by clicking on the main men "Scheduler"→"New"

A scheduler operates as follows (Fig. 36)

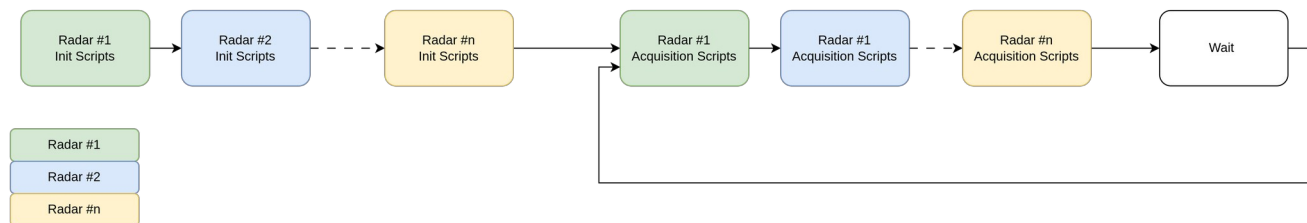


Fig. 36: Scheduler sequence of operations

As a first step the scheduler connects all required radar devices. When the connection is established for all intended devices, it runs all the init-scripts, in the saved order.

Then, it performs the radar cycle for all radar devices and their acquisition scripts are executed sequentially. If those cycle is less than the scheduler frame-rate a delay is inserted to meet frame-rate requirements.

The editor window for the scheduler is in Fig. 37. It includes the list of all devices included into the project along with check-boxes to include or exclude a certain radar in the scheduler cycle.

The scheduler has four main parameters:

1. Frame-rate (it is the time required by the scheduler to complete a new cycle).
2. Serial Timeout (it is the time required by any serial port to complete each transfer).
3. On Error Policy: if an error occur during scheduler cycle for a certain radar, one in four possible action is set:

- Halt All Devices: all devices are halted and the scheduler cycle stops
- Halt Device: halt only the device that caused the error. All other devices (if included) continue normally
- Continue: continue "normally" for all devices included in the scheduler operations
- Reinit Device: re-init the device that caused the error and then proceed normally with the scheduler cycle.

4. On Timeout Policy: if the execution cycle is longer than the Frame Rate, the scheduler

- Continue with next cycle ("Continue with available frame rate")
- Halt all devices

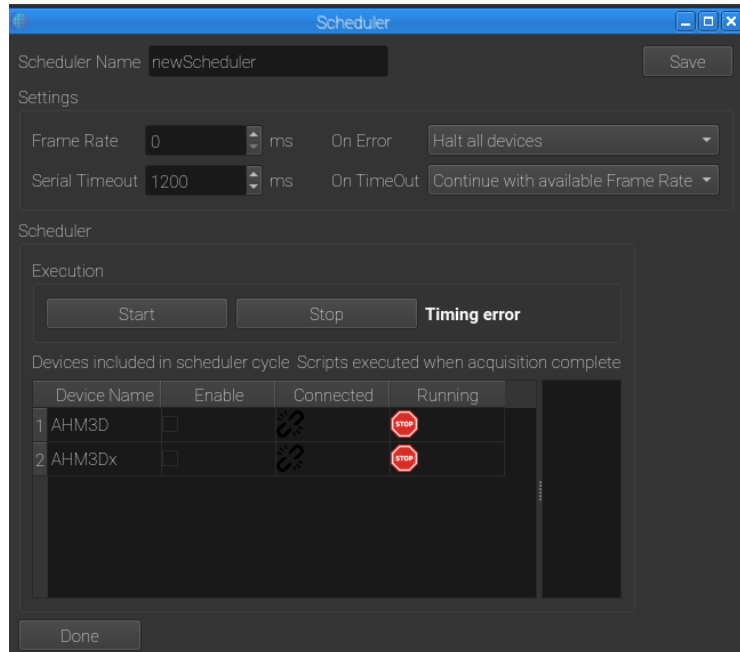


Fig. 37: Scheduler Window

NOTE: the ARIA-RDK is not a strict real-time software. Phase synchronization is not guaranteed among different radar devices.

The scheduler editor shows also the status of each radar (connected/disconnected, running/halted).

10.1 Tips

If a more precise synchronization among different radars is required, you may retrieve radar data from all of them in a single script that is executed as first "acquisition" script of the first radar of the scheduler. E.g.

```
var_immediate_inquiry("radar1_dataout");
var_immediate_inquiry("radar2_dataout");
```

```
....  
var_immediate_inquiry("radar#n_dataout");
```

11 Antennas

To Be Completed

12 Advanced Capabilities

To Be Completed

12.1 C/C++ scripts

When prototyping algorithms it may be useful, as an intermediate step before porting the algorithms to the radar module microprocessor (if present), to translate the algorithm in a C++ program that can be compiled to be executed as any other Octave script (see *Octave API* for reference. See https://docs.octave.org/v4.2.2/Matrices-and-Arrays-in-Oct_002dFiles.html#Matrices-and-Arrays-in-Oct_002dFiles for more details.

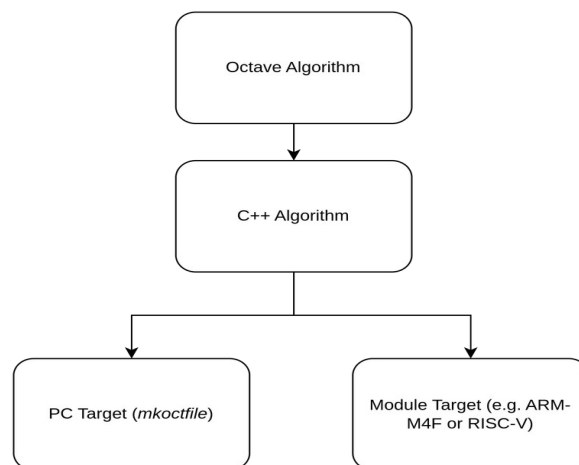


Fig. 38: Flow to bring algorithms from PC to module Microprocessor

A wrapper library to transfer a script in C++ written for Octave to a target microprocessor is under writing.



ARIA Sensing - Doc
20241001_DS_ARIARDK_v1p0
ARIA Sensing srl

12.2 ARIA UWB-Toolbox

To Be Completed

13 License

ARIA-RDK is distributed with a LGPL-3.0 License.

ARIA-RDK can be is distributed for the evaluation of algorithms based on ARIA UWB Radars SoCs and Modules. It can be used freely, but WITHOUT ANY WARRANTY; without even the implied warranty of FITNESS FOR A PARTICULAR PURPOSE.