

## Getting started with STM32CubeWiSE - Radio code generator

### Introduction

This document describes the STM32CubeWiSE-RadioCodeGenerator (STM32CubeWiSEcg) SW package with the STM32WL3x MRSUBG sequencer code generator.

STM32CubeWiSE-RadioCodeGenerator is a PC application that is used to build a flowgraph that defines which transceiver actions to execute under which condition, using the MRSUBG sequencer driver.

STM32WL3x Sub-GHz radio contains this sequencer, which is a state-machine like mechanism which allows for autonomous management of RF transfers, without any need for CPU intervention. If CPU intervention is required, interrupts can be defined. Transceiver actions can be arranged in a flowgraph. In this document, the individual transceiver actions are referred to as *SeqActions*.

However, source code is not the best representation for flowgraphs, since it conceals their logical and temporal structure. STM32CubeWiSE-RadioCodeGenerator addresses this issue by providing a graphical method to build flowgraphs, and then exporting the generated flowgraphs as C source code for integration into user applications.

The flowgraph definition is stored in the microcontroller RAM in the form of:

- A set of *ActionConfiguration* RAM tables, linked to each other using pointers. These pointers define the *SeqActions*, that is, the type of action (for example, transmission, reception, abort), as well as *SeqAction*-specific radio parameters and conditions for action transmissions.
- A unique *GlobalConfiguration* RAM table. This defines the entry point of the flowgraph (the first *SeqAction* to execute), as well as some default flag values and common radio parameters.

Radio parameters, which can be configured individually for each *SeqAction*, are stored in one of the dynamic registers, whose contents are part of the *ActionConfiguration* RAM table. Radio parameters that are fixed over the whole execution of the flowgraph (unless they are modified during a CPU interrupt), are stored in static registers, whose contents are part of the *GlobalConfiguration* RAM table.



## 1 General information

### 1.1 Licensing

This document describes software that runs on the STM32WL3x Arm® Cortex®-M0+ based microcontroller. For more information on the Arm® Cortex®-M0+, refer to the technical reference manuals, available from [ww.arm.com](http://ww.arm.com) website.



*Note: Arm and Cortex are registered trademarks of Arm Limited (or its subsidiaries or affiliates) in the US and/or elsewhere.*

*The Arm word and logo are trademarks of Arm Limited (or its subsidiaries) in the US and/or elsewhere. All rights reserved.*

### 1.2 Related documents

**Table 1. Document references**

Number	Reference	Title
[1]	RM0511	STM32WL30xx/31xx/33xx Arm®-based wireless MCUs with sub-GHz radio solution
[2]	RM0551	STM32WL3Rxx Arm®-based wireless MCUs with sub-GHz radio solution

## 2 Getting started

This section describes all of the system requirements to run STM32CubeWiSE-RadioCodeGenerator. It also details the software-package installation procedure.

### 2.1 System requirements

The STM32CubeWiSE-RadioCodeGenerator application has the following minimum setup requirements.

#### Supported operating systems and architectures

- Windows® 11, 64 bits (x64)
- Linux® Ubuntu® LTS 24.04

*Note:* *Windows is a trademark of the Microsoft group of companies.*  
*Linux® is a registered trademark of Linus Torvalds.*  
*Ubuntu® is a registered trademark of Canonical Ltd.*

### 2.2 STM32CubeWiSE-RadioCodeGenerator SW package setup

#### 2.2.1 Windows®

##### Install

1. Extract the content of the stm32wise-cgwin.zip file into a temporary directory, double click the installer file and follow the on-screen instructions.

##### How to launch

- Launch the STM32CubeWiSE-RadioCodeGenerator.exe file available on installation folder (app subfolder)

##### Uninstall

- Go to the STM32CubeWiSE-Radio Code Generator installation folder or to the Windows menu and select uninst000.exe

#### 2.2.2 Linux®

##### Install

Follow these instructions:

1. `sudo apt update`
2. `cp stm32cubewiseradiocodegenerator_VX.Y.Z_amd64.deb /tmp/`
3. `chmod a+r /tmp/stm32cubewiseradiocodegenerator_VX.Y.Z_amd64.deb`
4. `sudo apt install /tmp/stm32cubewiseradiocodegenerator_VX.Y.Z_amd64.deb`

The tool is located at `/local/home/{username}/ST`

##### How to launch

1. Type `stm32cubewiseradiocodegenerator`
2. After the first run from the command line, the application can be launched using the related installed tool icon

##### Uninstall

Follow these instructions:

1. `stm32cubewiseradiocodegenerator --reset`
2. `sudo apt autoremove stm32cubewiseradiocodegenerator`

On Linux® OS, the modemmanager process checks the COM ports when the board is plugged in. Due to this activity, the proper tool functionality is impacted:

- A specific rule is added at tool installation time, for allowing to filter modemmanager process on Nucleo kits COM ports which expose ST VID.
- As further action, user could deactivate the modemmanager activity before launching the tool with a specific instruction:
  - `sudo systemctl stop ModemManager`
  - The modemmanager activity can be restored with a specific instruction:
    - `sudo systemctl start ModemManager`

On Linux® OS, COM port could be not visible or connectable:

- The user might not have the proper access rights for ttyACM. Ubuntu requires adding the user to the dialout group with the command:
  - `sudo adduser username dialout command`

*Note:* Replace *username* with *user name*.

## 2.3 STM32CubeWiSE-RadioCodeGenerator SW package files

The STM32CubeWiSE-RadioCodeGenerator SW package files are organized into the following folders:

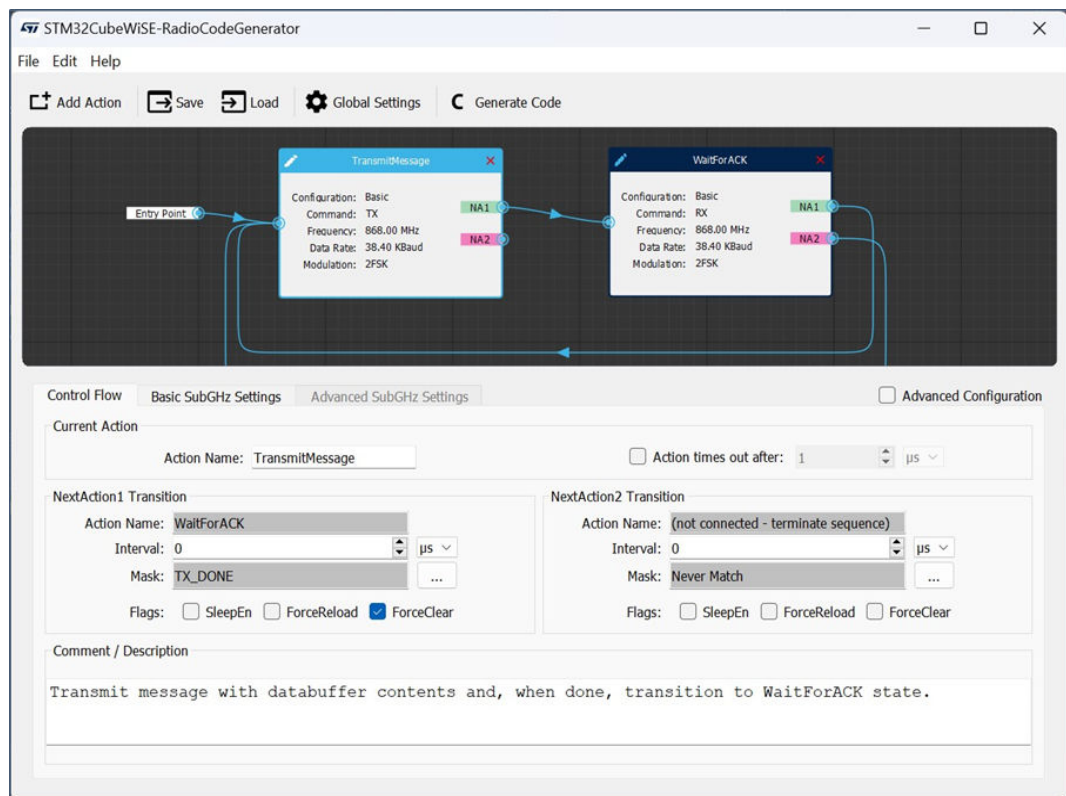
- **app**: contains the application launcher file.
- **examples**: this folder is organized into the following subfolders:
  - **code**: this folder contains the flowgraphs example already exported as C code, ready to be injected into an application project
  - **flowgraphs**: this folder stores some examples scenario of autonomous MRSUBG sequencer operations

Release notes and license files are located in the root folder.

### 3 STM32CubeWiSE-RadioCodeGenerator software description

This section describes the main functions of the STM32CubeWiSE-RadioCodeGenerator application. To run this utility, click on the STM32CubeWiSE-RadioCodeGenerator icon.

Figure 1. Main application window of STM32CubeWiSE-RadioCodeGenerator



DT58550V1

After launching STM32CubeWiSE-RadioCodeGenerator, the main application window appears. It consists of:

- A global menu and toolbar
- The visual drag-and-drop representation of the flowgraph
- The *SeqAction* configuration section (only visible if a *SeqAction* is currently being edited)

## 3.1 Building a flowgraph

### 3.1.1 Basics

Flowgraphs are built in two steps:

1. Add *SeqActions* to the flowgraph. This can be done using the “Add Action” button in the toolbar, using the global menu (Edit → Add Action) or with the “Ctrl+A” shortcut.
2. Connect *SeqActions* to the entry point and to each other by drawing action transition arrows.

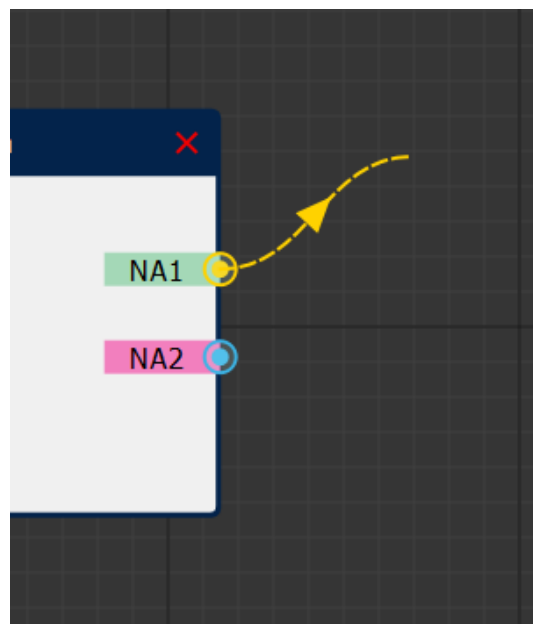
The conditions under which these transitions occur are defined later (see [Section 3.2.1: Control flow](#)).

### 3.1.2 Navigating the flowgraph, dragging actions

By dragging the checkerboard background of the flowgraph with the mouse pointer (left click), the viewport on the flowgraph can be adjusted. The mouse scroll wheel can be used to zoom in and out. Clicking anywhere on an action (except for the output ports, the delete button and the edit button) to select an action. Actions can be arranged in the flowgraph by dragging them with the left mouse button.

### 3.1.3 Adding action transitions

Figure 2. Drawing a transition for NextAction1



DT56551V1

As shown in [Figure 2](#), each action has two “output port”, called *NextAction1* (NA1) and *NextAction2* (NA2), which can be connected to *SeqActions* that are executed after the action has completed. For example, *NextAction1* could be used to execute some action if the current action was successful and *NextAction2* could be triggered in case of failure.

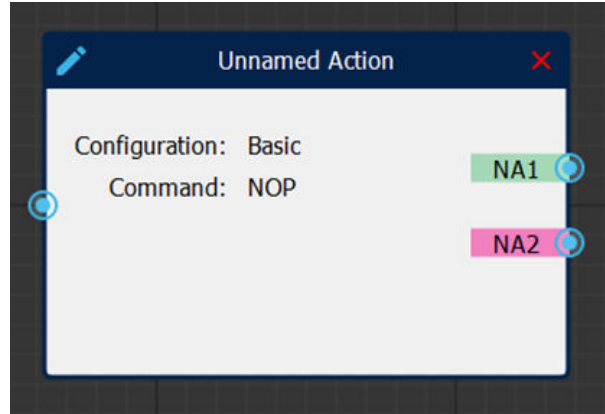
To create an action transition, hover the mouse pointer over one of the output ports, press the left mouse button and move the mouse pointer to drag a transition arrow. Move the mouse pointer over the input port on the left of some other *SeqAction* and release the left mouse button to make the connection permanent. To remove an action transition, just repeat the steps for creating an action transition, but release the left mouse button somewhere over the checkerboard background.

If an output (*NextAction1*, *NextAction2*) is left unconnected, the sequencer terminates if this next action is triggered.

Make sure to also connect the “Entry Point” to some *SeqAction*’s input port. This *SeqAction* is the first to be executed as soon as the sequencer is triggered.

### 3.1.4 Editing and deleting actions

Figure 3. SeqAction with edit button and delete button



DT58552V1

*SeqActions* can be edited by clicking on the pencil button on the top left of *SeqAction*. It can be deleted by clicking on the red cross on the top right (see Figure 3). Deleting a *SeqAction* also removes any incoming and outgoing action transitions.

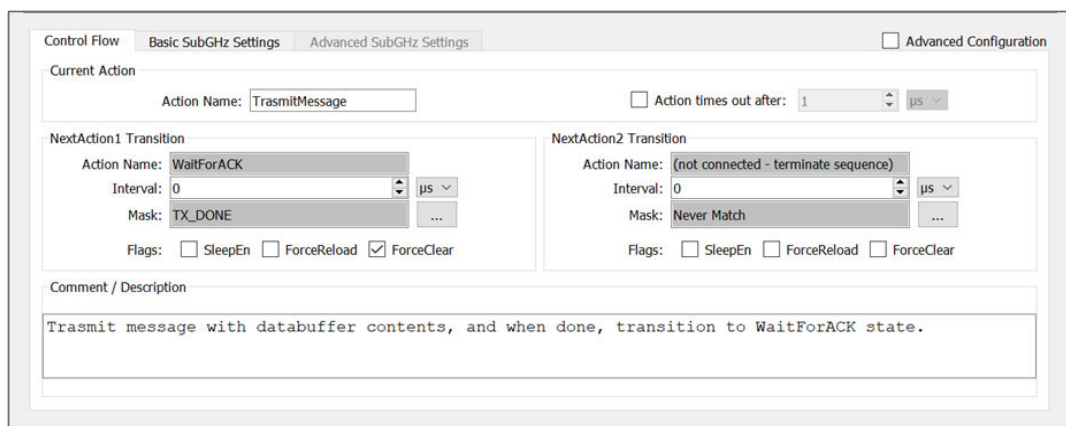
## 3.2 SeqAction configuration

*SeqActions* can be configured through a tabbed configuration interface accessible through the pencil button on the top left of each action in the flowgraph. This interface essentially configures the contents of the *ActionConfiguration* RAM table for the particular action, consisting of both control flow-related configuration options as well as the dynamic register contents. The dynamic register contents can either be configured manually with complete control over every register value (see Section 3.2.3: Advanced radio configuration) or through a simplified interface (see Section 3.2.2: Basic radio configuration). The simplified interface should be sufficient for almost all use cases.

### 3.2.1 Control flow

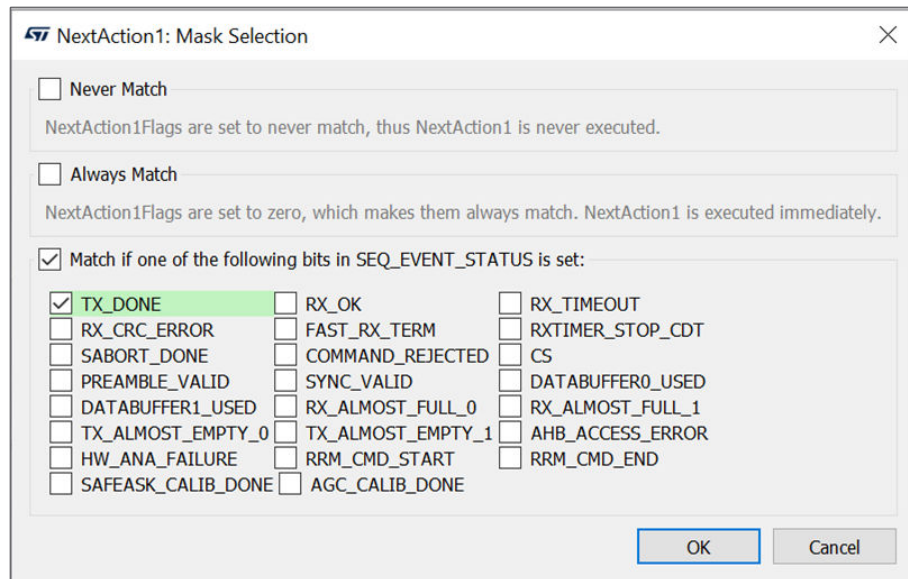
The control flow tab (see Figure 4) contains some basic configuration options such as action name and action timeout interval. The action name is not only used for display in the flowgraph, but is also carried over to the generated source code.

Figure 4. SeqAction control flow configuration tab



DT58553V1

Figure 5. Match mask selection for NextAction1, only one flag (TX\_DONE) is selected here



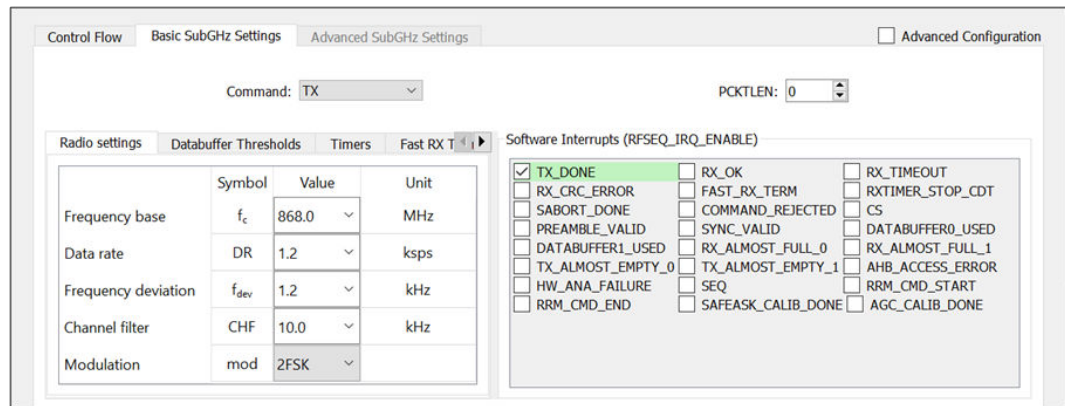
DT58554V1

The control flow tab (see Figure 4) contains some basic configuration options such as action name and action timeout interval. The action name is not just used for display in the flowgraph, but also carried over to the generated source code.

Most importantly, the control flow tab configures the condition on which a transition to NextAction1 / NextAction2 depends on as well as transition interval and flags. The transition condition can be configured by clicking on the button labelled "...", which makes the mask selection dialog shown in Figure 5 appear. The transition interval modified the NextAction1Interval / NextAction2Interval property of the RAM table. Refer to the STM32WL3x reference manual [1] for more information on the meaning of this interval and the significance of the *SleepEn* / *ForceReload* / *ForceClear* flags.

Furthermore, a short description of the *SeqAction* block can be added on this tab. This description is only used for documentation purposes and carried over to the generated source code as a source code comment.

### 3.2.2 Basic radio configuration

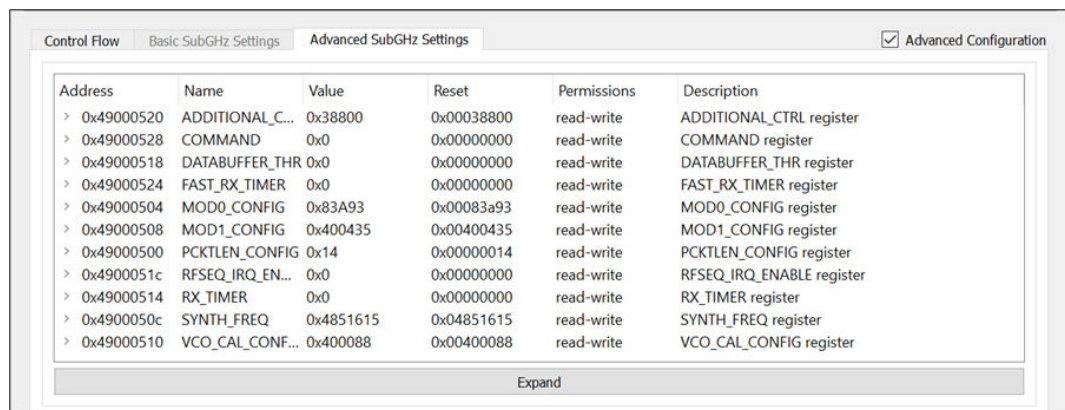
**Figure 6. Basic SubGHz settings tab**


DT58555V1

The basic radio configuration tab can be subdivided into three parts:

1. A section at the top where two of the most important parameters of any action are configured: the command to execute (TX, RX, NOP, SABORT, and so on) and, if applicable, the length of the packet to transfer.
2. A section on the left where the actual radio parameters such as: carrier frequency, data rate, modulation properties, data buffer thresholds and timers are configured.
3. A section on the right where the CPU interrupts can be individually enabled. An interrupt handler is generated for each of the ticked interrupts. This basically configures the contents of the RFSEQ\_IRQ\_ENABLE register. Refer to the STM32WL3x reference manual [1] for the meaning of the various radio parameters.

### 3.2.3 Advanced radio configuration

**Figure 7. Advanced radio configuration tab**


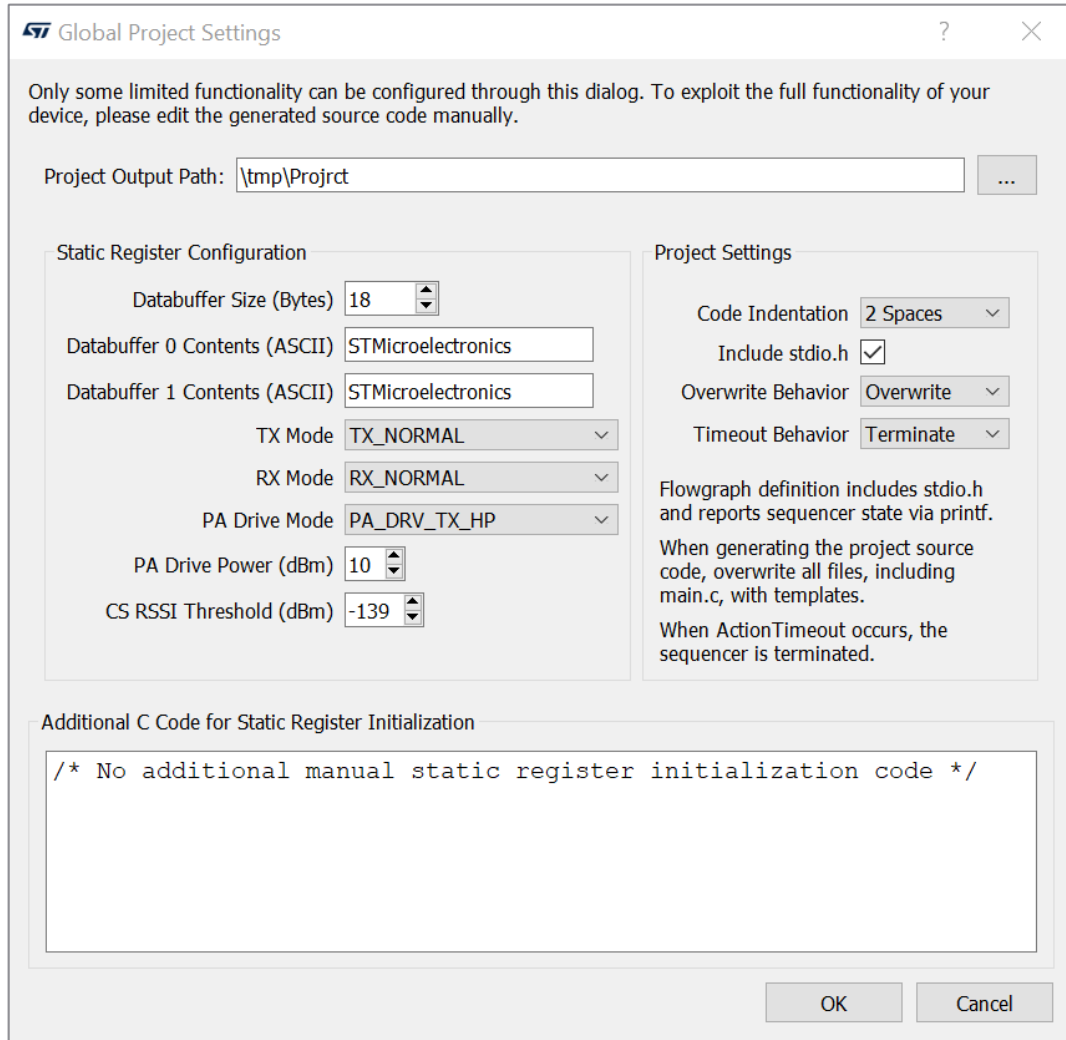
DT58556V1

If the configuration options exposed through the basic radio configuration tab (Section 3.2.2: Basic radio configuration) are insufficient, the advanced STM32WL3x radio configuration tab allows the setting of arbitrary dynamic register contents. The advanced configuration tab is enabled by ticking the *Advanced Configuration* checkbox to the top right of the tabbed configuration interface.

It is not possible to use both basic and advanced configuration at the same time, the user must select one or the other. However, it is of course also possible to manually edit the generated source code afterwards and to add potentially missing configuration options.

### 3.3 Global configuration dialog

Figure 8. Global project settings dialog with example values



DT56557V1

The “Global Project Settings” dialog can be accessed through the “Global Settings” toolbar button. The dialog contains both configuration options for the static register contents as well as additional project settings. Note that only a small fraction of static register configuration options can be configured through this dialog. These options are only provided to speed up application prototyping applications with STM32CubeWiSE-RadioCodeGenerator. It is usually expected that the static register contents are set up in the application’s manually-written source code. The meaning of the other project settings is explained in the dialog itself.

Additional C code that is inserted just before creating the *Global Configuration* RAM table from the static register contents may also be provided. This field may be used to set up static register values which are inaccessible through the provided static register configuration mask.

### 3.4 Code generation

The flowgraph can be translated into a complete project C source code by pressing the *Generate Code* button in the toolbar. The generated project folder does not contain project files for IAR, Keil®, or GCC. These files must be added manually to the STMWL3x project.

This is the generated project folder structure:

- Project folder
  - *inc*
    - *SequencerFlowgraph.h*: header file for *SequencerFlowgraph.c*, static. Do not edit this.
    - *stm32wl3x\_hal\_conf.h*: STM32WL3x HAL configuration file, static.
  - *src*
    - *SequencerFlowgraph.c*: flowgraph definition. This is the important file that uses the sequencer driver to define the global-configuration and action-configuration RAM tables. Autogenerated, do not edit.
    - *main.c*: Project main file that demonstrates how to load and apply the flow-graph definition. Static, modify this as needed.

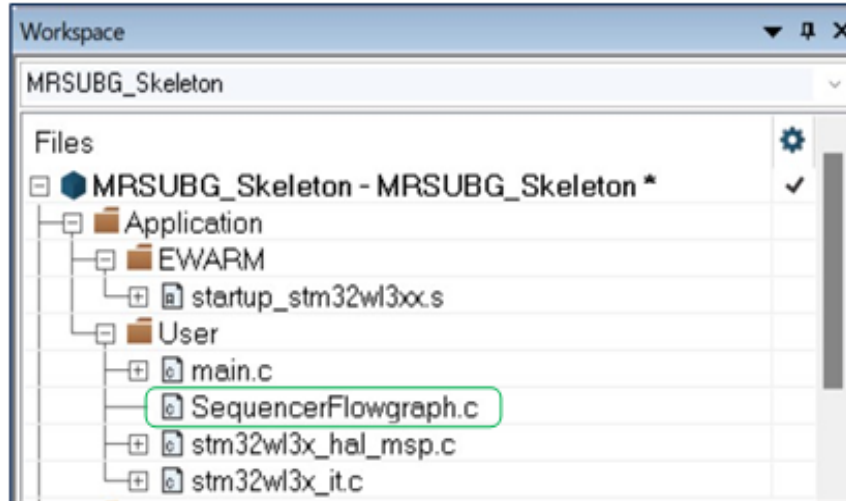
To edit *main.c* or *stm32wl3x\_hal\_conf.h*, select overwrite behavior *Keep* in the project settings. This way, only *SequencerFlowgraph.c* gets overwritten.

### 3.5 How to import generated code into a CubeMX project for STM32CubeWL3

To import a project generated by STM32CubeWiSE-RadioCodeGenerator into a CubeMX project, it is necessary to follow the following steps:

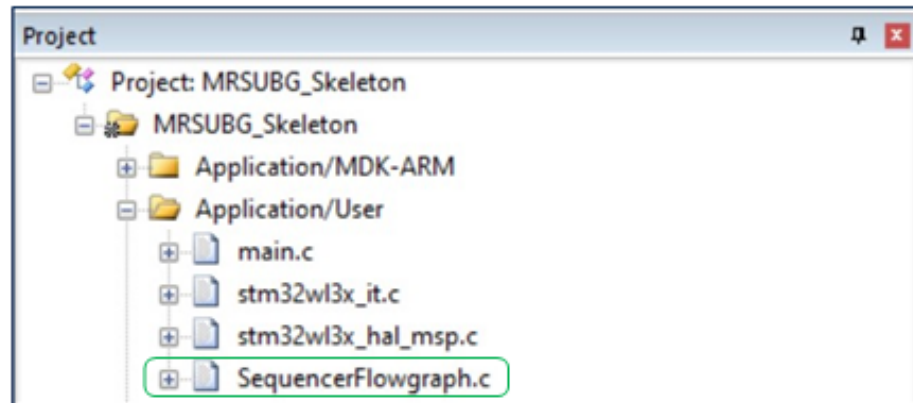
1. Open the folder contained the files generated by STM32CubeWiSE-RadioCodeGenerator and copy “*Inc*” and “*Src*” folders.
2. Paste the two folders on “*MRSUBG\_Skeleton*” folder, located at *Examples\MRSUBG* under *Firmware\Projects\NUCLEO-WL33CC* or *NUCLEO-WL3RKB1* or *NUCLEO-WL3*, overwriting the two already present.
3. Open “*MRSUBG\_Skeleton*” project in one of the following IDEs:
  - EWARM
  - MDK-ARM
  - STM32CubeIDE

4. Inside the “MRSUBG\_Skeleton” project, add “SequencerFlowgraph.c” file:
  - For an EWARM project, the path to add the file is the following: *MRSUBG\_Skeleton\Application\User*

**Figure 9. EWARM project**


DT58561V1

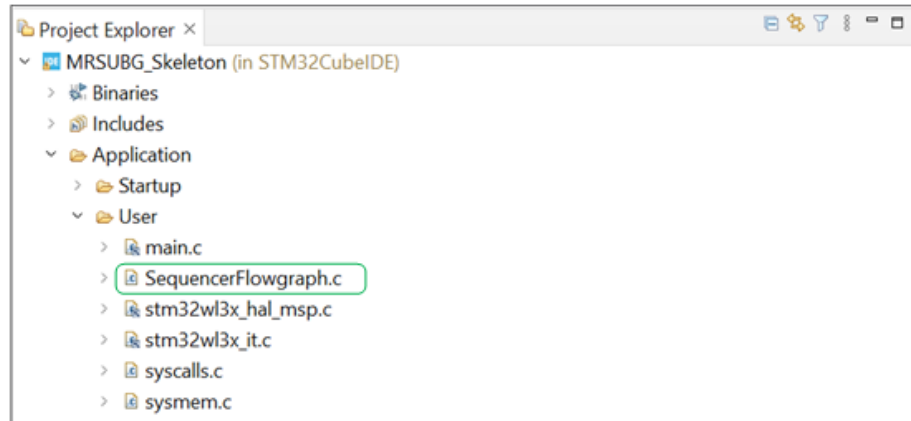
- For an MDK-ARM project, the path to add the file is the following: *MRSUBG\_Skeleton\Application\User*

**Figure 10. MDK-ARM project**


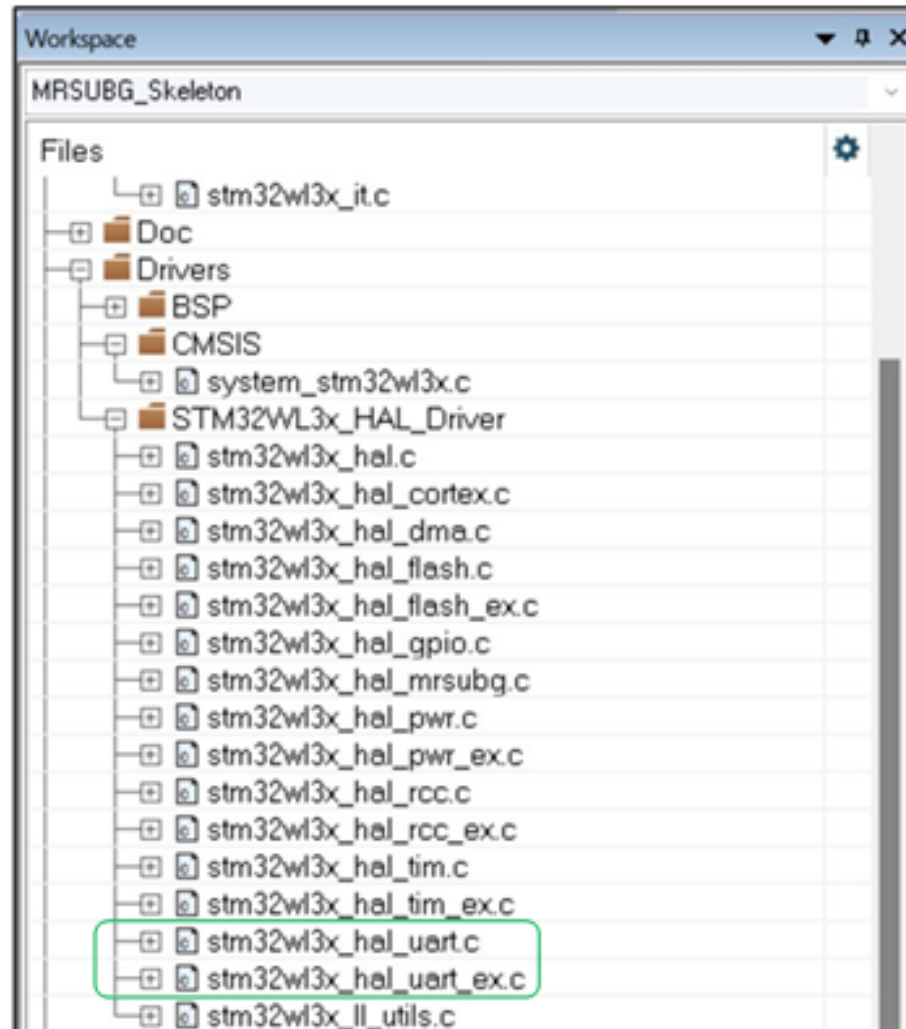
DT58562V1

- For an STM32CubeIDE project, the path to add the file is the same:  
*MRSUBG\_Skeleton\Application\User*

Figure 11. STM32CubeIDE project



5. Inside the *MRSUBG\_Skeleton* project, add *stm32wl3x\_hal\_uart.c* and *stm32wl3x\_hal\_uart\_ex.c* files to the following path: *MRSUBG\_Skeleton\Drivers\STM32WL3x\_HAL\_Driver*. The path is the same for all IDEs. The two files are located at *Firmware\Drivers\STM32WL3x\_HAL\_Driver\Src*.

**Figure 12. Drivers folder for EWARM project**


6. To use COM features, *stm32wl3x\_nucleo\_conf.h* file, located at *Examples\MRSUBG\MRSUBG\_Skeleton\Inc* under *Firmware\Projects\NUCLEO-WL33CC* or *NUCLEO-WL3RKB1* or *NUCLEO-WL3RKB2*, must be modified setting *USE\_BSP\_COM\_FEATURE* and *USE\_COM\_LOG* to 1U:

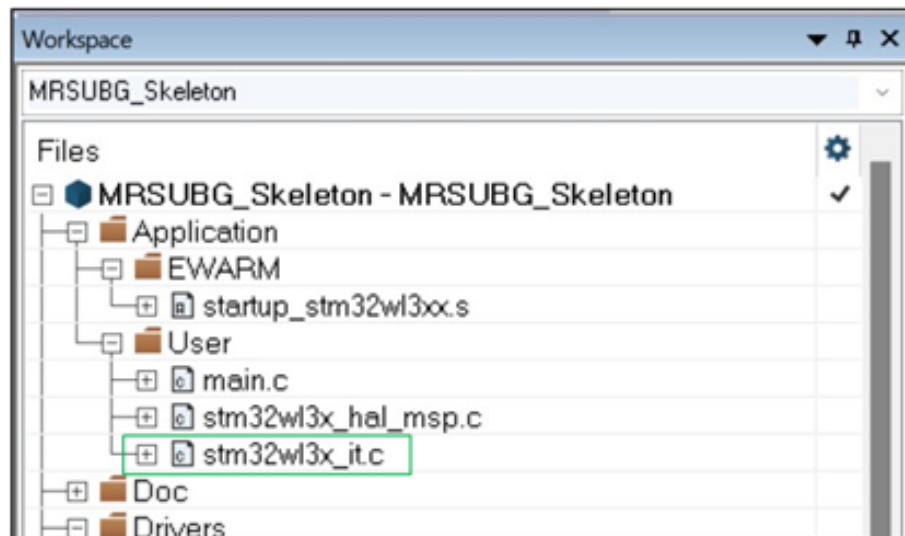
**Figure 13. stm32wl3x\_nucleo\_conf.h file**

```

29
30  /* Usage of nucleo board */
31  #define USE_NUCLEO_64      1U
32
33  /* Usage of COM feature */
34  #define USE_BSP_COM_FEATURE 1U
35  #define USE_COM_LOG        1U
36
37  /* Button interrupt priorities */
38  #define BSP_B1_IT_PRIORITY 0x0FUL /* Default is lowest priority level */
39  #define BSP_B2_IT_PRIORITY 0x0FUL /* Default is lowest priority level */
40  #define BSP_B3_IT_PRIORITY 0x0FUL /* Default is lowest priority level */
41
    
```

DT58565V2

7. Copy the following code into "*stm32wl3x\_it.c*", located at *MRSUBG\_Skeleton\Application\User*.

**Figure 14. Application section for EWARM project**


DT58566V1

```

/** @brief This function handles MRSUBG interrupt. */
void MRSUBG_IRQHandler(void)
{
    /* USER CODE BEGIN MRSUBG_IRQn 0 */

    /* USER CODE END MRSUBG_IRQn 0 */
    HAL_MRSUBG_IRQHandler();
    /* USER CODE BEGIN MRSUBG_IRQn 1 */

    /* USER CODE END MRSUBG_IRQn 1 */
}
    
```

8. Add the following function prototype into *stm32wl3x\_it.h*: `void MRSUBG_IRQHandler(void);`.

9. Uncomment the HAL\_UART\_MODULE\_ENABLED define in *stm32wl3x\_hal\_conf.h*, located at *Examples\MRSUBG\MRSUBG\_Skeleton\Inc* under *Firmware\Projects\NUCLEO-WL33CC* or *NUCLEO-WL3RKB1* or *NUCLEO-WL3RKB2*.

**Figure 15. HAL\_UART\_MODULE\_ENABLED define to delete**

```

42  /*#define HAL_I2S_MODULE_ENABLED */
43  /*#define HAL_IRDA_MODULE_ENABLED */
44  /*#define HAL_IWDG_MODULE_ENABLED */
45  /*#define HAL_RNG_MODULE_ENABLED */
46  /*#define HAL_RTC_MODULE_ENABLED */
47  /*#define HAL_SMARTCARD_MODULE_ENABLED */
48  /*#define HAL_SMBUS_MODULE_ENABLED */
49  /*#define HAL_SPI_MODULE_ENABLED */
50  /*#define HAL_TIM_MODULE_ENABLED */
51  #define HAL_UART_MODULE_ENABLED
52  /*#define HAL_USART_MODULE_ENABLED */
53  /*#define HAL_WWDG_MODULE_ENABLED */
54  /*#define HAL_CRYPT_MODULE_ENABLED */
55  /*#define HAL_LCD_MODULE_ENABLED */
56  /*#define HAL_LCSC_MODULE_ENABLED */
57  /*#define HAL_LPAUR_MODULE_ENABLED */
58  #define HAL_MRSUBG_MODULE_ENABLED
59  /*#define HAL_MRSUBG_TIMER_MODULE_ENABLED */
60  /*#define HAL_DAC_MODULE_ENABLED */
61

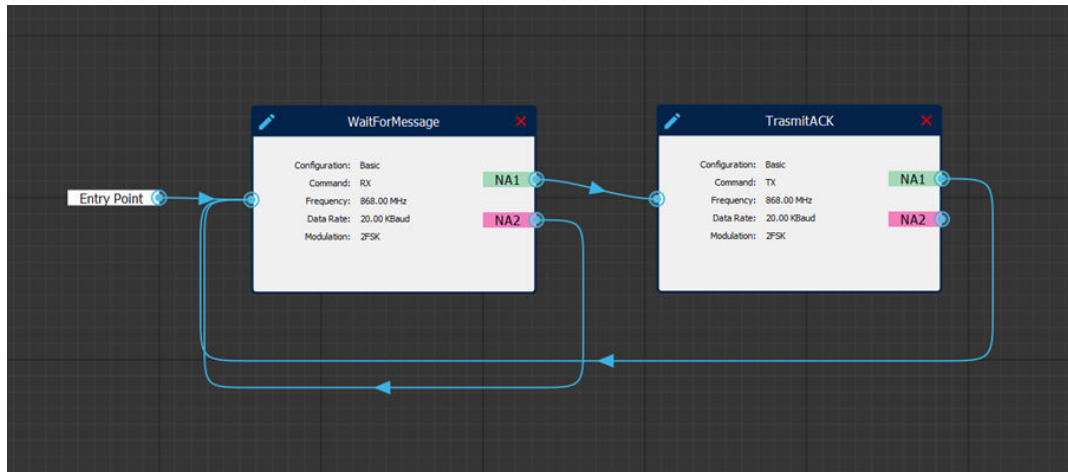
```

## 3.6 Flowgraph examples

Four example flowgraphs are provided alongside the source code. These examples may be loaded into STM32CubeWiSE-RadioCodeGenerator by clicking the “Load” button in the toolbar.

### 3.6.1 AutoACK\_RX

Figure 16. AutoACK\_RX example



DT58559V1

The *Auto-ACK* demo illustrates how two STM32WL3x devices can automatically talk to each other with minimal CPU intervention, with the help of the sequencer hardware.

This flowgraph implements the behavior (*Auto-Transmit-ACK*) of device A. In device A, the sequencer is initialized in a receiving state (*WaitForMessage*), in which it waits for a message to arrive.

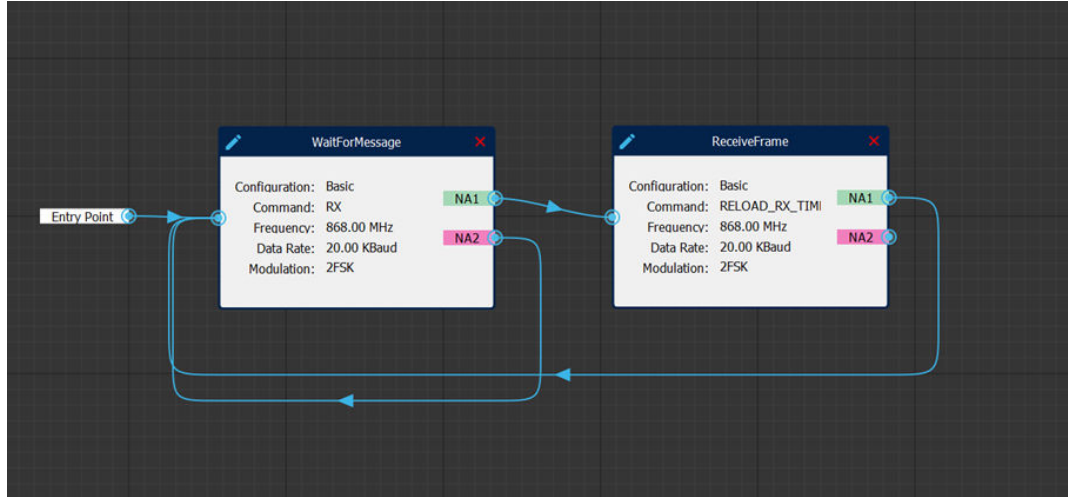
Once a valid message arrives, the sequencer automatically transitions into an transmit state (*TransmitACK*), in which an ACK packet is sent as a response, without CPU intervention. Once this is finished, the sequencer is reset into its initial *WaitForMessage* state.

This flowgraph implements the same behavior as the *MRSUBG\_SequencerAutoAck\_Rx* example from the *Examples\MRSUBG* folder of the STM32Cube WL3 Software package. If *AutoACK\_RX* is flashed on one device, A, and *AutoACK\_TX* is flashed on some device, B, the two devices send messages back and forth, as in a ping-pong game.



3.6.4 Sniff mode

Figure 19. Sniff mode example



DT58569V1

This example is taken from the STM32WL3x reference manual [1]. Refer to that manual for further details of this example.

## Revision history

**Table 2. Document revision history**

Date	Version	Changes
21-Nov-2024	1	Initial release.
10-Feb-2025	2	Updated device name to scope STM32WL3x.
02-Apr-2025	3	Updated Section 3.5: How to import generated code into a CubeMX project for STM32CubeWL3, including: <ul style="list-style-type: none"> <li>• <i>MRSUBG_Skeleton</i> folder pathname</li> <li>• Updated Figure 13. <i>stm32wl3x_nucleo_conf.h</i> file</li> <li>• New step 8</li> <li>• New Figure 15. <i>HAL_UART_MODULE_ENABLED</i> define to delete</li> </ul>
05-Nov-2025	4	Updated Table 1. Document references
17-Feb-2026	5	Updated Section 3.5: How to import generated code into a CubeMX project for STM32CubeWL3
29-Jun-2026	6	Updated title. Updated Section 2.1: System requirements and Section 2.2: STM32CubeWiSE-RadioCodeGenerator SW package setup. Added Section 2.2.1: Windows® and Section 2.2.2: Linux®. Updated Section 2.3: STM32CubeWiSE-RadioCodeGenerator SW package files.

## Contents

<b>1</b>	<b>General information</b>	<b>2</b>
1.1	Licensing	2
1.2	Related documents	2
<b>2</b>	<b>Getting started</b>	<b>3</b>
2.1	System requirements	3
2.2	STM32CubeWiSE-RadioCodeGenerator SW package setup	3
2.2.1	Windows®	3
2.2.2	Linux®	3
2.3	STM32CubeWiSE-RadioCodeGenerator SW package files	5
<b>3</b>	<b>STM32CubeWiSE-RadioCodeGenerator software description</b>	<b>6</b>
3.1	Building a flowgraph	7
3.1.1	Basics	7
3.1.2	Navigating the flowgraph, dragging actions	7
3.1.3	Adding action transitions	7
3.1.4	Editing and deleting actions	8
3.2	SeqAction configuration	8
3.2.1	Control flow	8
3.2.2	Basic radio configuration	10
3.2.3	Advanced radio configuration	10
3.3	Global configuration dialog	11
3.4	Code generation	12
3.5	How to import generated code into a CubeMX project for STM32CubeWL3	12
3.6	Flowgraph examples	18
3.6.1	AutoACK_RX	18
3.6.2	AutoACK_TX	19
3.6.3	Listen before talk (LBT)	19
3.6.4	Sniff mode	20
	<b>Revision history</b>	<b>21</b>

## List of tables

<b>Table 1.</b>	Document references . . . . .	2
<b>Table 2.</b>	Document revision history . . . . .	21

## List of figures

<b>Figure 1.</b>	Main application window of STM32CubeWiSE-RadioCodeGenerator. . . . .	6
<b>Figure 2.</b>	Drawing a transition for NextAction1 . . . . .	7
<b>Figure 3.</b>	SeqAction with edit button and delete button . . . . .	8
<b>Figure 4.</b>	SeqAction control flow configuration tab. . . . .	8
<b>Figure 5.</b>	Match mask selection for NextAction1, only one flag (TX_DONE) is selected here. . . . .	9
<b>Figure 6.</b>	Basic SubGHz settings tab. . . . .	10
<b>Figure 7.</b>	Advanced radio configuration tab . . . . .	10
<b>Figure 8.</b>	Global project settings dialog with example values . . . . .	11
<b>Figure 9.</b>	EWARM project . . . . .	13
<b>Figure 10.</b>	MDK-ARM project. . . . .	13
<b>Figure 11.</b>	STM32CubeIDE project . . . . .	14
<b>Figure 12.</b>	Drivers folder for EWARM project . . . . .	15
<b>Figure 13.</b>	stm32wl3x_nucleo_conf.h file . . . . .	16
<b>Figure 14.</b>	Application section for EWARM project . . . . .	16
<b>Figure 15.</b>	HAL_UART_MODULE_ENABLED define to delete . . . . .	17
<b>Figure 16.</b>	AutoACK_RX example . . . . .	18
<b>Figure 17.</b>	AutoACK_TX example. . . . .	19
<b>Figure 18.</b>	Listen before talk example . . . . .	19
<b>Figure 19.</b>	Sniff mode example . . . . .	20

**IMPORTANT NOTICE – READ CAREFULLY**

STMicroelectronics NV and its subsidiaries (“ST”) reserve the right to make changes, corrections, enhancements, modifications, and improvements to ST products and/or to this document at any time without notice.

In the event of any conflict between the provisions of this document and the provisions of any contractual arrangement in force between the purchasers and ST, the provisions of such contractual arrangement shall prevail.

The purchasers should obtain the latest relevant information on ST products before placing orders. ST products are sold pursuant to ST’s terms and conditions of sale in place at the time of order acknowledgment.

The purchasers are solely responsible for the choice, selection, and use of ST products and ST assumes no liability for application assistance or the design of the purchasers’ products.

No license, express or implied, to any intellectual property right is granted by ST herein.

Resale of ST products with provisions different from the information set forth herein shall void any warranty granted by ST for such product.

If the purchasers identify an ST product that meets their functional and performance requirements but that is not designated for the purchasers’ market segment, the purchasers shall contact ST for more information.

ST and the ST logo are trademarks of ST. For additional information about ST trademarks, refer to [www.st.com/trademarks](http://www.st.com/trademarks). All other product or service names are the property of their respective owners.

Information in this document supersedes and replaces information previously supplied in any prior versions of this document.

© 2026 STMicroelectronics – All rights reserved