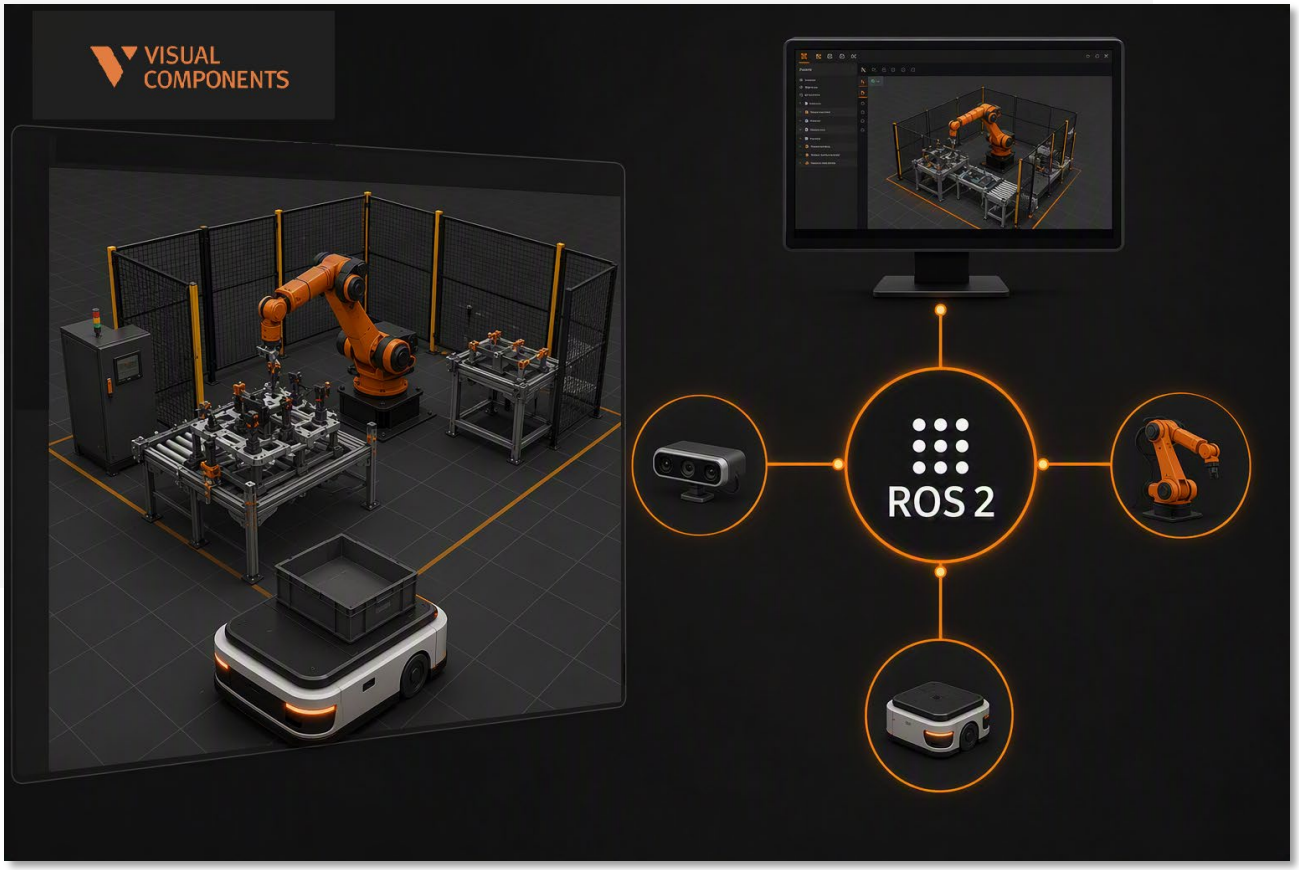
Visual Components ROS2 Connection User Guide

Visual Components 5.2 | Version: November 12, 2026



This tutorial guides you through establishing a communication channel between Visual Components 5.2 and ROS2 on Windows.

You will learn how to start VisualComponents ROS2 Bridge, create topics and message payload and see how to move messages between Visual Components and ROS2 ecosystem.

Contents

Scope and Audience 6
What this guide covers 6
What this guide does not cover 6
Prerequisites and Required Files 7
Package file structure 8
Tested Environment and Reference Paths 9
Step 1: Initialize ROS2 11
1.1 Confirm ROS2 and Pixi workspace paths 11
1.2 Start the Pixi command environment 11
1.3 Load the ROS2 setup file 11
1.4 Verify ROS2 is available 11
Step 2: Verify the VCROS2Connection Package 12
2.1 Extract the bridge package 12
2.2 Verify the required bridge files 12
2.3 Update paths if your folder is different 12
Step 3: Start the Bridge for Manual Testing 12
3.1 Start the bridge for manual testing 12
3.2 Select the correct ROS2 distribution 13
3.3 Keep the bridge running during ROS2 visibility testing 13
Step 4: Verify the Bridge Node in ROS2 13
4.1 Check the bridge node from ROS2 13
4.2 Confirm that ROS2 can see the bridge 13
Step 5: Test ROS2 Topics with the Keyboard Tester 14
5.1 Start the keyboard tester 14
5.2 Disable odometry display if required 14
5.3 Confirm that the command velocity topic is available 14
5.4 Check the command velocity topic details 15
Step 6: Confirm ROS2 Message Transmission 16
6.1 Echo the command velocity topic 16
6.2 Send a movement command from the keyboard tester 16

6.3 Confirm that message values change 16
6.4 Stop the test processes 16
Step 7: Prepare for VC Runtime Testing 17
Step 8: Run the Bridge from VC 17
8.1 Copy the VC command files 17
8.2 Start VC 5.2 application 17
8.3 Open the ROS2 Config command 17
8.4 Open the demo layout 19
8.5 Start the keyboard tester 20
8.6 Start the VC simulation 21
1. Start the simulation in VC. 21
2. Wait 10–20 seconds for the first bridge startup. 21
3. Confirm that a new command window opens with bridge session information. This guide refers to it as Window4. 21
4. Check the VC Output panel. For the demo layout, no error messages should appear. 21
8.7 Understand reset behavior 21
8.8 Confirm bridge and keyboard tester processes 22
8.9 Test robot movement 22
Troubleshooting and FAQ 24
Appendix A: XML Pairing 30
Full example 30
Elements and attributes 30
Direction and packet types 32
Configuration errors 32
Appendix B: DataPackets 32
Dynamic custom packets 33
Appendix C: Supported ROS2 Message Types 33
std_msgs 33
geometry_msgs 34
nav_msgs 34

sensor_msgs 34
trajectory_msgs 35

Before you start, review the scope, audience, and required setup for this guide.

Terminology:
VCROS2Connection is the feature and the name of the distributed ZIP package and extracted folder.

The package contains the bridge executables **vc_ros2_bridge_jazzy.exe** and **vc_ros2_bridge_lyrical.exe**, plus the configuration, launcher, test, command, icon, and demo files.

In this guide, **the bridge** means the executable selected for the chosen ROS2 distribution; **run_bridge.py** starts the appropriate executable.

Scope and Audience

Technical audience: This guide assumes that the reader is already familiar with the ROS2 ecosystem, Windows command-line usage, ROS2 topics and nodes, and basic Python programming. This requirement describes the user's technical knowledge, not the Visual Components product edition.

What this guide covers

This guide explains how to use the provided VCROS2Connection package to validate ROS2 communication, test topic publishing, and connect Visual Components simulation runtime with a ROS2 environment.

What this guide does not cover

- Installing ROS2 on Windows.
- Installing Pixi or creating the Pixi workspace.

1. Installing Python or configuring Python on PATH.

- Teaching ROS2 concepts such as nodes, topics, publishers, subscribers, and message types.

2. Explaining Python programming or modifying the provided Python scripts.
3. Building ROS2 from source or compiling custom ROS2 packages.

Install ROS2, Pixi, and Python using their official documentation before following the connection validation steps in this guide.

NOTE: The connection maps configured **ROS 2 topics** for the message types in Appendix C;it **does not connect ROS2 services or actions** and is not a general ROS graph bridge.

MoveIt, ros2_control, and Nav2 are not integrated with VC through this guide's setup.

VC Product Compatibility:

VCROS2Connection runtime is not restricted to a specific Visual Components product edition.

A prepared layout that already contains the required DataPacket behaviours and stored ROS2 configuration can be run in **Visual Components Essentials, Professional, or Premium**, provided that the VCROS2Connection bridge package is available.

Creating or modifying the required DataPacket behaviours requires **Professional or Premium**.

Prerequisites and Required Files

4. Visual Components 5.2 Essentials, Professional, or Premium.
   Essentials requires a prepared layout with the required DataPackets and stored ROS2 configuration.

- Windows machine with ROS2 installed.
  This guide expects an existing ROS2 Jazzy or Lyrical installation and does not cover ROS2 installation.
- Pixi installed and available from Command Prompt.
  Pixi installation and workspace creation are not covered in this guide.
- Python installed and available from Command Prompt.
  Python installation and PATH configuration are not covered in this guide.

5. VCROS2Connection folder extracted to a path without spaces.

- VC and the bridge executable must run on the same Windows PC.
  Running VC on the host PC and bridge executable inside a virtual PC is not currently supported for VC-to-ROS2 communication.

6. The ZIP file must contain the files listed in the package structure below.

Package file structure

After extracting the ZIP file, the VCROS2Connection folder should contain the following 10 files:

Do not rename the files names

File	Purpose	Used in
MobileRobot_ros2_config.xml	XML pairing file that maps ROS2 topics to VC signals.	Loaded in the ROS2 Config command before running the simulation.
vc_ros2_bridge_jazzy.exe	Bridge executable for ROS2 Jazzy.	Selected automatically by run_bridge.py when --distro jazzy is used.
vc_ros2_bridge_lyrical.exe	Bridge executable for ROS2 Lyrical.	Selected automatically by run_bridge.py when --distro lyrical is used.
WheelDrive_ROS_demo.vcmx	VC demo layout configured for ROS2 bridge testing.	Opened in VC for runtime testing.
keyboard_tester.py	Publishes keyboard-based velocity commands and can read odometry.	Used by run_keyboard_tester.py during ROS2 topic testing.
run_bridge.py	Starts the correct bridge executable for the selected ROS2 distribution.	Used in manual bridge testing and bridge startup validation.
run_keyboard_tester.py	Starts the keyboard tester with the selected ROS2 distribution.	Used to publish /myrobot/cmd_vel during ROS2 topic testing.

runtime.py	Starts the bridge during VC simulation runtime.	Used when VC starts the bridge automatically.
cmdVCROS2Connection.py	VC Python command for loading the ROS2 XML configuration.	Copied to the VC 5.2 commands folder and used from the Home Tab.
VCROS2Connection.svg	Icon for the ROS2 Config command	Copied to the VC 5.2 commands folder

Note:

- VCROS2Connection currently supports ROS2 Jazzy and ROS2 Lyrical.
- cmdVCROS2Connection.py is a VC Python command.

7. The XML pairing file is required. It maps ROS2 topics to VC signals. Without it, the bridge can start and ROS2 topics can exist, but messages will not be transmitted to or from VC.

Tested Environment and Reference Paths

This setup was tested with the following folders and ROS2 distribution:

ROS2 Pixi location check:

Before continuing, confirm that the ROS2 Pixi installation is available in one of the locations checked by **runtime.py**.

By default, the script uses **C:\pixi_ws**.
For Lyrical, it also detects **D:\pixi_ws\ros2-windows** and then uses **D:\pixi_ws** as the Pixi workspace.

8. Preferred distribution-specific installation: **<Pixi workspace>\ros2-jazzy** or **<Pixi workspace>\ros2-lyrical**, when that folder contains **local_setup.bat**.
   - Fallback installation: **<Pixi workspace>\ros2-windows**.
9. The script also requires **pixi.toml** and the Pixi environment folder **.pixi\envs\default\Library\bin** in the workspace or ROS2 installation folder.

If ROS2 is not installed in either supported location: update the location used in **runtime.py** before starting the bridge. In particular, update the default Pixi workspace or ROS2 installation path so that it points to the folder containing **local_setup.bat**, **pixi.toml**, and the Pixi environment.

Alternatively, define **VC_ROS2_PIXI_WS** and, when needed, **VC_ROS2_INSTALL** with the correct paths before launching Visual Components.

Setting	Reference value
VCROS2Connection folder	`C:\VCROS2Connection`
ROS2 install	`D:\pixi_ws\ros2-windows`
ROS2 setup file	`D:\pixi_ws\ros2-windows\local_setup.bat`
Pixi workspace	`D:\pixi_ws`
ROS2 distro	`lyrical`

Use these paths for the commands in this guide. If your installation uses different folders, replace the paths with your own.

Important: Each bridge executable can be tested independently from VC, but when used with VC simulation runtime, the selected executable must run on the same Windows PC as VC. Communication between the VC application and the bridge is currently supported only on the same computer.

Step 1: Initialize ROS2

1.1 Confirm ROS2 and Pixi workspace paths

Before starting the bridge, confirm where ROS2 and the Pixi workspace are installed so the commands and **runtime.py** point to the correct folders.

Verify that **local_setup.bat**, **pixi.toml**, and **.pixi\envs\default\Library\bin** exist in the resolved workspace or ROS2 installation. If the installation is outside the default **C:\pixi_ws** location and the Lyrical fallback **D:\pixi_ws\ros2-windows**, update the paths in **runtime.py** or set **VC_ROS2_PIXI_WS** and **VC_ROS2_INSTALL** before continuing.

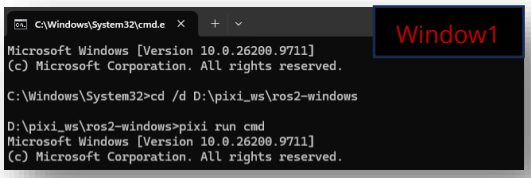
This test uses the example paths below. Replace the highlighted paths with the locations used on your computer:

```
ROS2_install:   D:\pixi_ws\ros2-windows
Pixi workspace: D:\pixi_ws
```

1.2 Start the Pixi command environment

Open Command Prompt (**Window1**) and run:

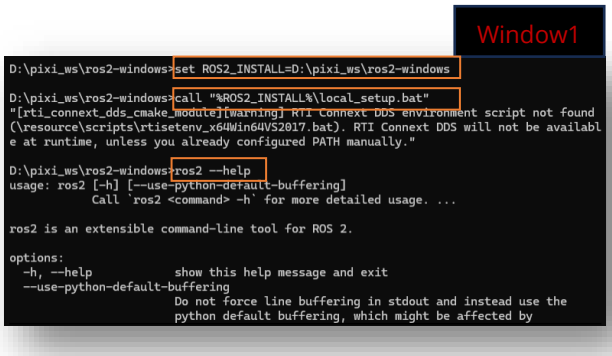
```
cd /d D:\pixi_ws\ros2-windows
pixi run cmd
```



1.3 Load the ROS2 setup file

In the same window, run:

```
set ROS2_INSTALL=D:\pixi_ws\ros2-windows
call "%ROS2_INSTALL%\local_setup.bat"
```



1.4 Verify ROS2 is available

Check that ROS2 is available:

```
ros2 --help
```

Expected result:

```
usage: ros2 ...
```

Step 2: Verify the VCROS2Connection Package

2.1 Extract the bridge package

Extract VCROS2Connection.zip to a folder whose path does not contain spaces. For example:

`C:\VCROS2Connection`

2.2 Verify the required bridge files

Do not place the package in C:\Program Files or another protected folder. Confirm that the extracted folder contains all files listed in the Package file structure table above.

Confirm that all files are present before continuing. Missing files can prevent the bridge, keyboard tester, VC command, or demo layout from working correctly.

2.3 Update paths if your folder is different

If your package is in another folder, replace,

`C:\VCROS2Connection`

in all commands in the following sections with your actual path.

Step 3: Start the Bridge for Manual Testing

3.1 Start the bridge for manual testing

Use this manual bridge window only to test the command and read bridge errors directly. Later, VC will start the bridge automatically during simulation runtime.

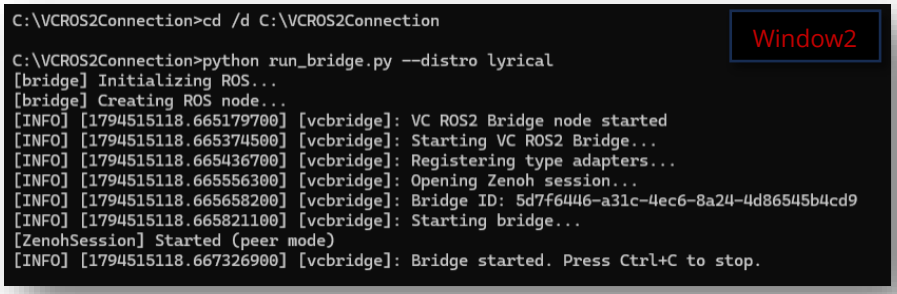
Open a second Command Prompt(**Window2**) and run:

```
cd /d C:\VCROS2Connection
```

```
python run_bridge.py --distro lyrical
```

Expected result:

```
[bridge] Initializing ROS...
[bridge] Creating ROS node...
[INFO] [1788896209.411806900] [vcbridge]: VC ROS2 Bridge node started
[INFO] [1788896209.412405700] [vcbridge]: Starting VC ROS2 Bridge..
```



3.2 Select the correct ROS2 distribution

If you are using jazzy as your ROS2 environment, pass *jazzy* to run the bridge

```
python run_bridge.py --distro jazzy
```

The launcher selects the matching executable automatically:

```
jazzy   -> vc_ros2_bridge_jazzy.exe
lyrical -> vc_ros2_bridge_lyrical.exe
```

3.3 Keep the bridge running during ROS2 visibility testing

Leave this command prompt open while you test ROS2 visibility.

Before starting the simulation in VC using the XML `bridgeLauncher`, stop the bridge you started in Step 3.1 by pressing `Ctrl+C` or closing the window so that VC can start the bridge automatically.

Step 4: Verify the Bridge Node in ROS2

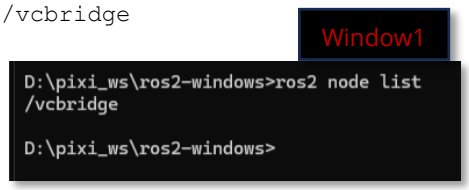
4.1 Check the bridge node from ROS2

Go back to **Window1** and run:

```
ros2 node list
```

Expected result includes:

```
/vcbridge
```



4.2 Confirm that ROS2 can see the bridge

If `/vcbridge` is listed, ROS2 can see the bridge.

Step 5: Test ROS2 Topics with the Keyboard Tester

5.1 Start the keyboard tester

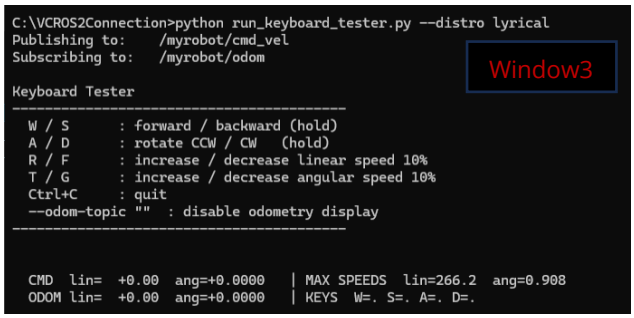
To test whether the bridge can publish and subscribe to messages, open a third Command Prompt(**Window3**) and run:

```
cd /d C:\VCROS2Connection
```

```
python run_keyboard_tester.py --distro lyrical
```

Expected result:

```
Publishing to:    /myrobot/cmd_vel
Subscribing to:   /myrobot/odom
```



5.2 Disable odometry display if required

If you do not have odometry yet, disable odometry display:

```
python run_keyboard_tester.py --distro lyrical --odom-topic=
```

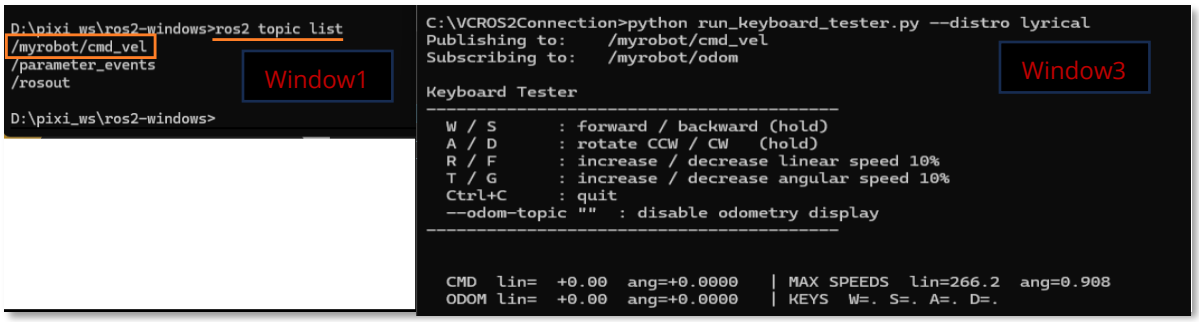
5.3 Confirm that the command velocity topic is available

Go back to **Window1** and run:

```
ros2 topic list
```

Expected result includes:

```
/myrobot/cmd_vel
```



If you do not see the expected result, make sure run_keyboard_tester is still running in Window3

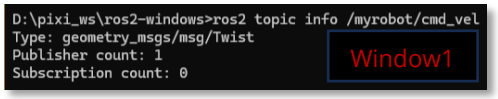
5.4 Check the command velocity topic details

Once you see the topic, check the topic type by running:

`ros2 topic info /myrobot/cmd_vel (in Window1)`

Expected result:

```
Type: geometry_msgs/msg/Twist
Publisher count: 1
```



NOTE:

Before VC starts using this topic, `Subscription count` can be `0`.

After VC starts the simulation and loads the subscriber mapping, run the same command again. Expected result then includes:

Subscription count: 1

If the subscription count stays `0`, VC has not connected this topic to the bridge yet.

Step 6: Confirm ROS2 Message Transmission

6.1 Echo the command velocity topic

In your **Window1**, run:

```
ros2 topic echo /myrobot/cmd_vel
```

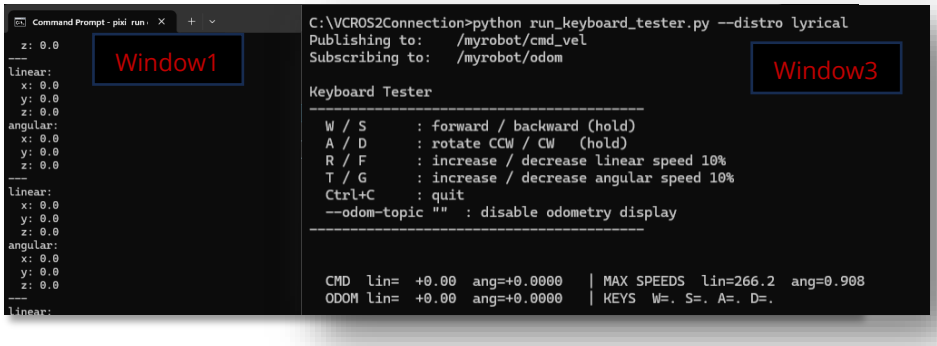
6.2 Send a movement command from the keyboard tester

Go to the **Window3** and hold `W`.

Expected result in the **Window1**:

```
linear:
x: 200.0

angular:
z: 0.0
```



6.3 Confirm that message values change

The exact number can be different if speed was changed. The important part is that `linear.x` changes when `W` or `S` is pressed, and `angular.z` changes when `A` or `D` is pressed.

6.4 Stop the test processes

Press Ctrl+C in **Window1** to stop ros2 topic echo, but keep **Window1** open.

Press Ctrl+C in **Window2** to stop the selected bridge executable.

Keep **Window1** and **Window3** open for VC runtime testing. Since VC will start the selected bridge executable automatically, you can close **Window2**.

Step 7: Prepare for VC Runtime Testing

After validating the bridge manually, use **Step 8** to start the selected bridge executable from VC and test message exchange during simulation runtime. Complete the earlier steps successfully before continuing. If issues occur, see the FAQ section.

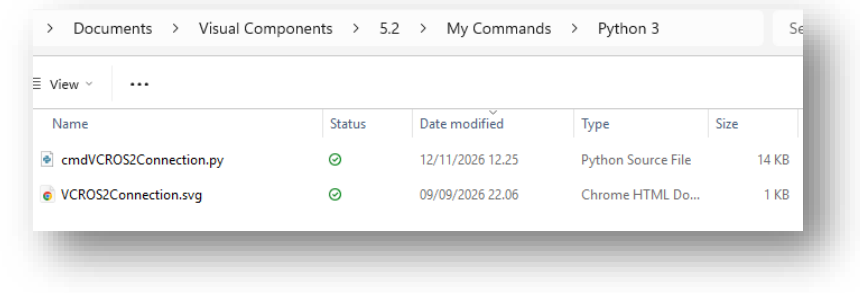
Step 8: Run the Bridge from VC

This section explains how to load the ROS2 configuration in VC, start the bridge from the simulation runtime, and test message exchange with the keyboard tester.

8.1 Copy the VC command files

Copy **cmdVCROS2Connection.py** and **VCROS2Connection.svg** from C:\VCROS2Connection to:

```
%USERPROFILE%\Documents\Visual Components\5.2\My Commands\Python
```

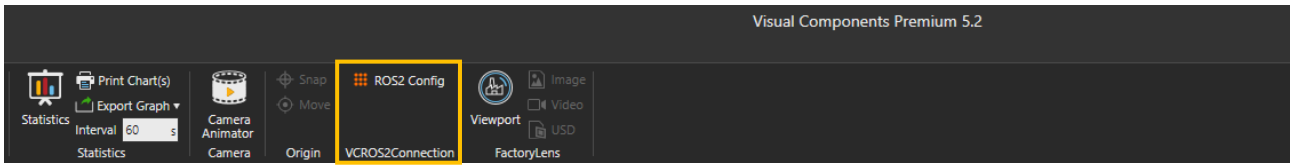


8.2 Start VC 5.2 application

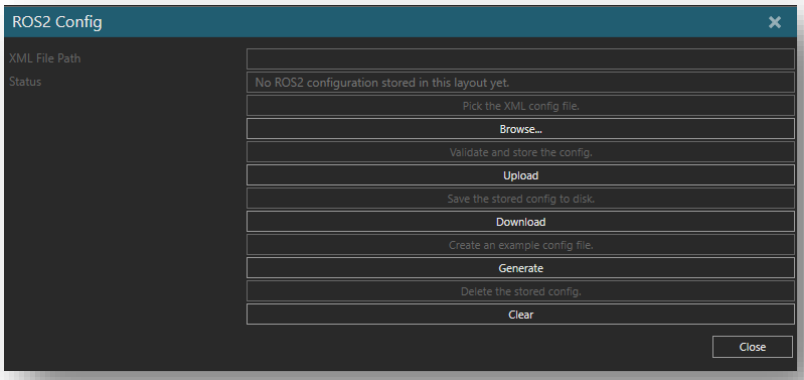
Launch Visual Components 5.2 and wait for the application to finish loading. Confirm that the Home tab is available before continuing.

8.3 Open the ROS2 Config command

ROS2 Config command should be available from
Home tab -> VCROS2Connection group.



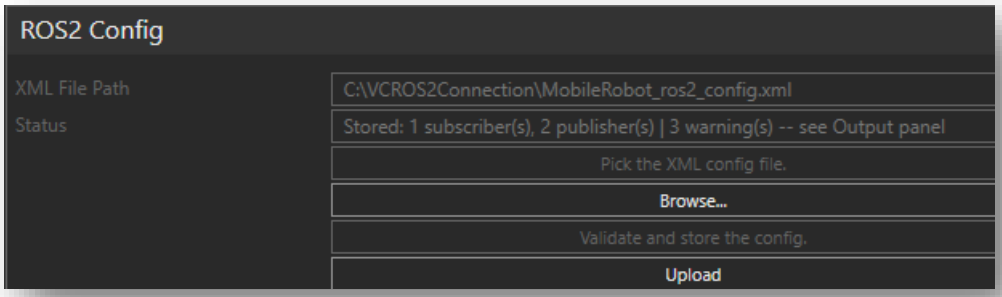
The ROS2 Config command manages the ROS2 Connectivity XML configuration stored with the current layout.
Select the command to see the following command properties



- **XML File Path** – Read-only field showing the selected configuration-file path. It is filled through **Browse…** or remembered from the last **Upload** or **Download**.
- **Status** – Read-only summary of the stored configuration, including subscriber and publisher counts or any validation errors and warnings.
- **Browse…** – Opens a file picker to select the XML configuration file.
- **Upload** – Validates the XML structure, required attributes, and signal existence, then stores the configuration in the current layout.
- **Download** – Writes the stored configuration to the XML File Path and asks for confirmation before overwriting an existing file.
- **Generate** – Creates a new example XML file at the XML File Path and asks for confirmation before overwriting an existing file.
- **Clear** – Deletes the stored configuration from the current layout after confirmation.
- **Close** – Closes the panel.

Load an XML configuration: Skip these steps for **WheelDrive_ROS_demo.vcmx**; its configuration is already loaded. For a new layout, follow the steps below.

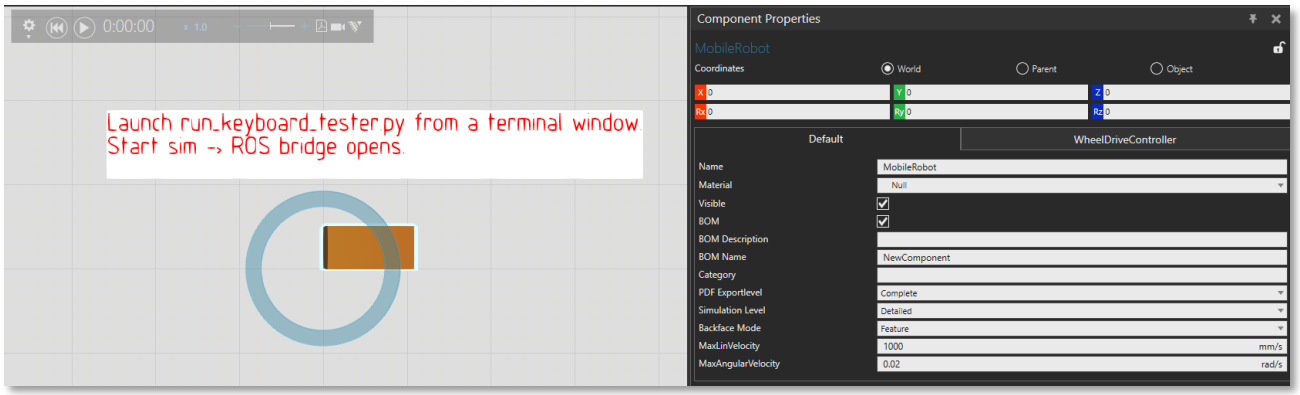
1. Select **Browse...**,
2. Choose the XML file, 'MobileRobot_ros2_config.xml' from the VCROS2Connection folder
3. Select **Upload**.
4. Review the read-only **Status** field for mapping counts, errors, or warnings before continuing.



8.4 Open the demo layout

Open the demo layout: **WheelDrive_ROS_demo.vcmx**.

The layout contains a **MobileRobot** component with bridge-related **WheelDriveController** and **DataPacket** behaviors.



The layout already contains the XML configuration.
To review it, open the **ROS2 Config** command and select **Download**.

The XML maps ROS2 topics to **MobileRobot** signals used by **WheelDriveController**. It receives velocity commands through **In_velocity** and publishes odometry and IMU data through **Out_odometry** and **Out_IMU**.

Subscriber

- Receives **/myrobot/cmd_vel** as **geometry_msgs/msg/Twist** and forwards velocity commands to **In_velocity**.

Publishers

- **Odometry:** Publishes **Out_odometry** to **/myrobot/odom** topic as **nav_msgs/msg/Odometry**, using frame IDs **odom** and **base_link**.
- **IMU:** Publishes **Out_IMU** to **/mobile_robot/imu** as **sensor_msgs/msg/Imu**, using frame ID **base_link**.

These mappings define how **WheelDriveController** exchanges data with ROS2.

8.5 Start the keyboard tester

In **Window3**, start the keyboard tester:

```
python run_keyboard_tester.py --distro lyrical
```

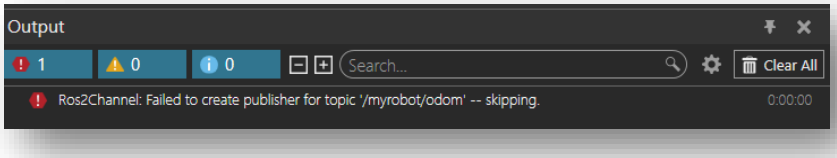
Confirm that it starts without errors.



8.6 Start the VC simulation

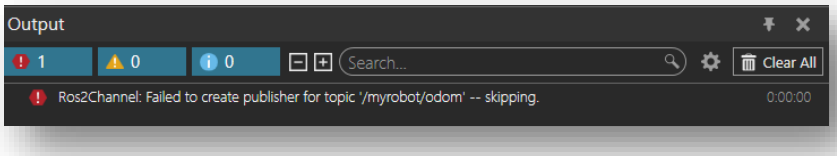
1. Start the simulation in VC.
2. Wait 10–20 seconds for the first bridge startup.
3. Confirm that a new command window opens with bridge session information. This guide refers to it as Window4.
4. Check the VC Output panel. For the demo layout, no error messages should appear.

If the bridge process does not start



verify the XML **bridgeLauncher** path, upload the corrected XML file, and start the simulation again. See **FAQ Q13** for the complete checks.

If VC cannot create a publisher or subscriber:



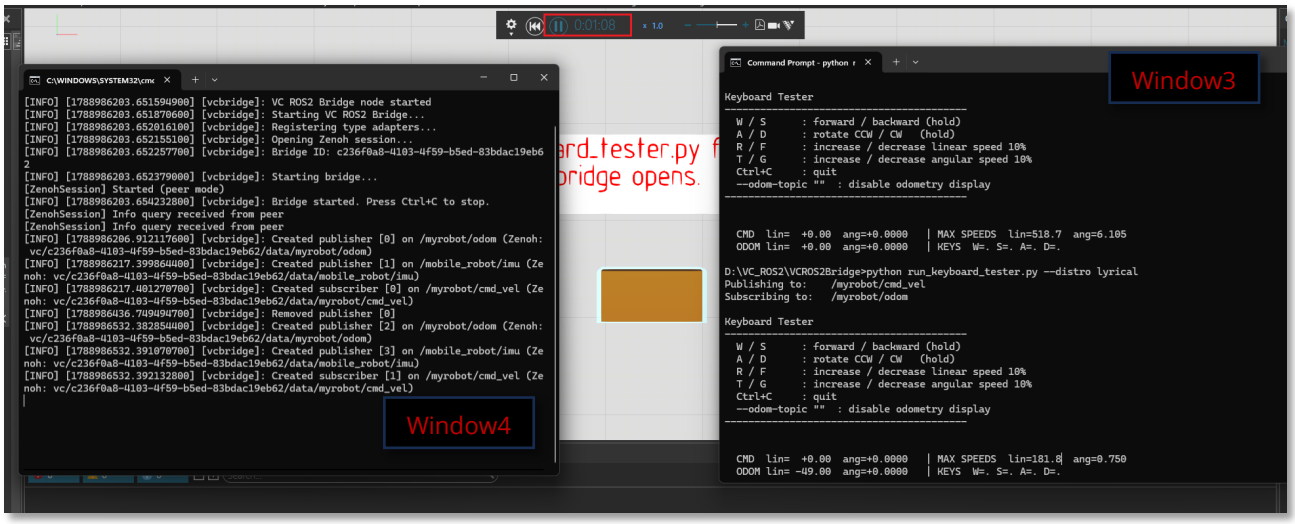
reset the simulation, save and reload the layout, and start the simulation again. See **FAQ Q14** for the complete recovery steps.

8.7 Understand reset behavior

When you reset the simulation, VC removes the publisher and subscriber mappings, but the selected bridge executable may keep running. Starting the simulation again recreates the mappings from the XML configuration.

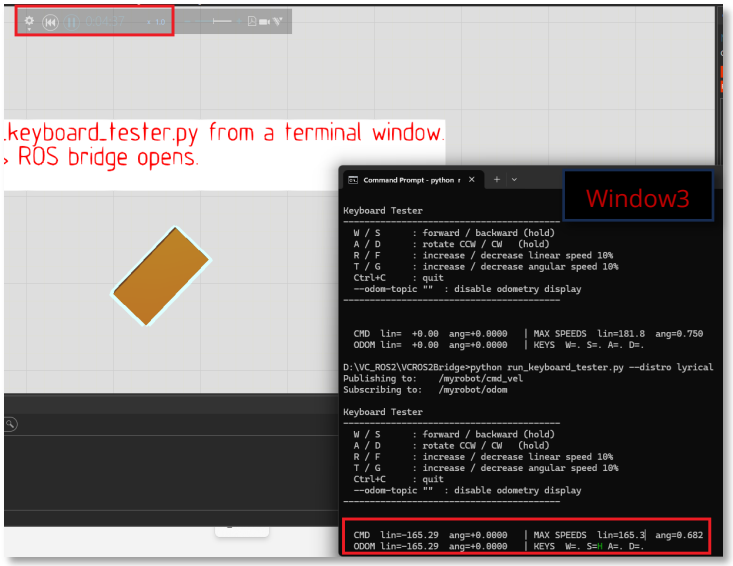
8.8 Confirm bridge and keyboard tester processes

Confirm that both the selected bridge executable (**vc_ros2_bridge_jazzy.exe** or **vc_ros2_bridge_lyrical.exe**) and **keyboard_tester** are running.



8.9 Test robot movement

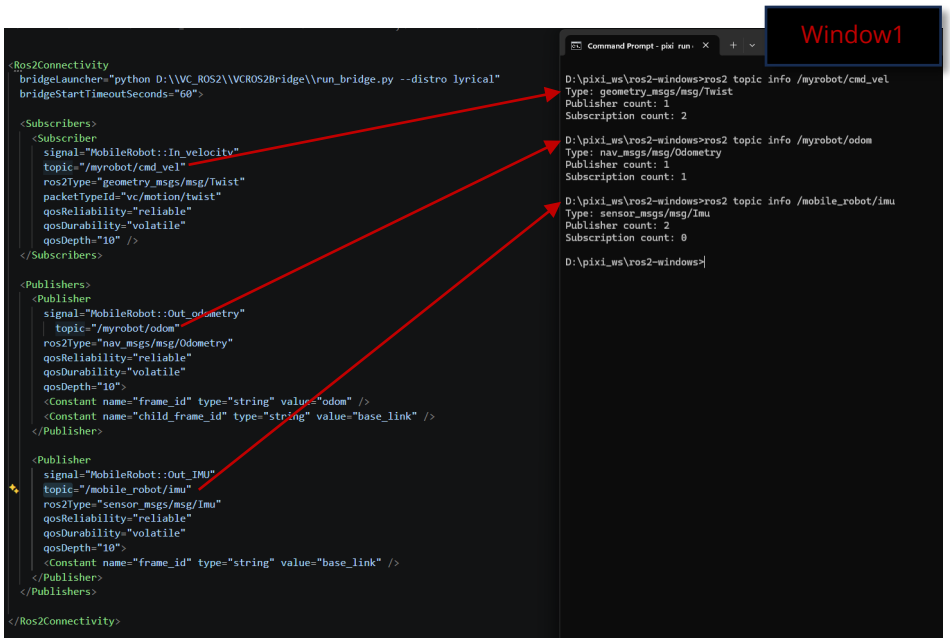
In **Window3**, hold **W**, **S**, **A**, or **D**. The **MobileRobot** should move in the 3D world.



If the robot does not move, check the topic mapping, publisher count, subscriber count, and message type from **Window1**.

Run these commands to check the topic details:

```
ros2 topic info /myrobot/cmd_vel
ros2 topic info /myrobot/odom
ros2 topic info /mobile_robot/imu
```



You can also echo a topic to confirm that ROS2 messages are being received.

When finished, close **Window1**, **Window3**, **Window4**, and VC to stop the test session cleanly.

This completes the basic VCROS2Connection runtime test.

Troubleshooting and FAQ

Q1. Is VC acting as an overlay?
A1. No. VC is not acting as a ROS2 overlay.

Think of the ROS2 folder as the installed ROS2 system. It contains ROS2 tools and standard ROS2 packages.

Think of VC as a separate application. It talks to ROS2 through the bridge program.

Because of this, `ros2 pkg list` shows installed ROS2 packages only. It will not show VC or the bridge.

Use this command to check the bridge instead:

```
ros2 node list
```

Expected result:

```
/vcbridge
```

Q2. Why do I not see anything related to VC when I run `ros2 pkg list`
A2. That is expected.

`ros2 pkg list` lists installed ROS2 packages. The VC bridge is a program, not a ROS2 package installed into the ROS2 folder.

Use these commands instead:

```
ros2 node list
ros2 topic list
```

Q3. Why do I see `/vcbridge` but not `/myrobot/imu`?
A3. The bridge is running, but VC has not created that topic yet.

1.Upload the XML pairing file in the ROS2 Config window

2.Start the VC simulation, and make sure the XML has a publisher for `/myrobot/imu`.

Q4. Why is the XML pairing file important?
A4. The XML pairing file tells VC what to do with ROS2 topics.

For example, it tells VC:

```
Receive /myrobot/cmd_vel and send it to MobileRobot::VelocityIn.
Publish MobileRobot::ImuOut as /myrobot/imu.
```

Without the XML file, the bridge can run and the keyboard tester can publish, but VC has no instruction to receive `/myrobot/cmd_vel`.

The robot will not move.

Q5. Why do I see `/myrobot/cmd_vel` but VC does not move?
A5. Check these items:

- The VC simulation is running.
- The XML pairing file was uploaded in the ROS2 Config command.
- The layout has a subscriber for `/myrobot/cmd_vel`.
- The subscriber uses message type `geometry_msgs/msg/Twist`.
- The subscriber is connected to the correct signal in the layout.

Also check the ROS2 subscription count:

```
ros2 topic info /myrobot/cmd_vel
```

Expected result while VC is connected:

```
Subscription count: 1
```

If it says `Subscription count: 0`, the keyboard tester is publishing, but VC is not receiving the command through the bridge.

Q6. Why does the keyboard tester show odometry as zero?
A6. The keyboard tester listens to `/myrobot/odom` by default.

If the VC layout does not publish `/myrobot/odom`, the odometry line will stay zero. This does not block keyboard command testing.

To hide odometry display, run:

```
python "%BRIDGE_HOME%\run_keyboard_tester.py" --distro %ROS2_DISTRO% --odom-topic=

BRIDGE_HOME=<full path to your VCROS2Connection folder>
ROS2_DISTRO=<jazzy or lyrical>
```

Q7. '/vcbridge' does not appear

A7. Check that the bridge window is still running.

Then run the following command in your Window1:

```
ros2 node list
```

If '/vcbridge' is still missing, stop the bridge with `Ctrl+C` in the command prompt, start it again, and check for error text in the bridge window.

Q8. `/myrobot/cmd_vel` does not appear

A8. Check that the keyboard tester is running.

Then run:

```
ros2 topic list
```

If the topic is still missing, restart the keyboard tester.

Q9. `/myrobot/cmd_vel` appears but subscription count is zero

A9. Check Subscription count from the ros2 topic info

Run:

```
ros2 topic info /myrobot/cmd_vel
```

If the output shows this:

```
Publisher count: 1
Subscription count: 0
```

then the keyboard tester is publishing, but VC has not connected to that topic.

Check these items in VC:

- The simulation is running.
- The XML pairing file was uploaded in the ROS2 Config command.
- The XML contains a subscriber for `/myrobot/cmd_vel`.
- The XML uses `ros2Type="geometry_msgs/msg/Twist"`.
- The XML points to the correct VC signal, for example `MobileRobot::VelocityIn`.

The bridge window should stay open while you do this check.

Q10. `ros2 topic echo /myrobot/cmd_vel` prints zeros only

A10. This is normal when no key is pressed.

Keep the echo command running, then hold `W`, `S`, `A`, or `D` in the keyboard tester window.

Q11. Can I run VC on the host PC and ROS2 in a virtual PC?
A11. No. For VC runtime communication, the selected bridge executable from the VCROS2Connection package must run on the same Windows PC as VC.

You can test the bridge executable independently from VC, but the communication between the VC application and the selected bridge executable is currently supported only on the same computer. Therefore, a setup where VC runs on the host PC and the bridge runs inside a virtual PC is not supported.

VC and bridge in different machines is unsupported

VC and bridge on Windows, with additional ROS 2 nodes in an Ubuntu VM is not tested

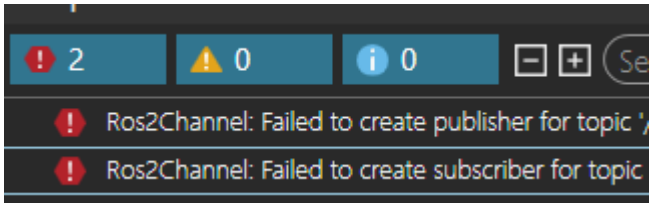
Q12. What should I check if the bridge does not start?

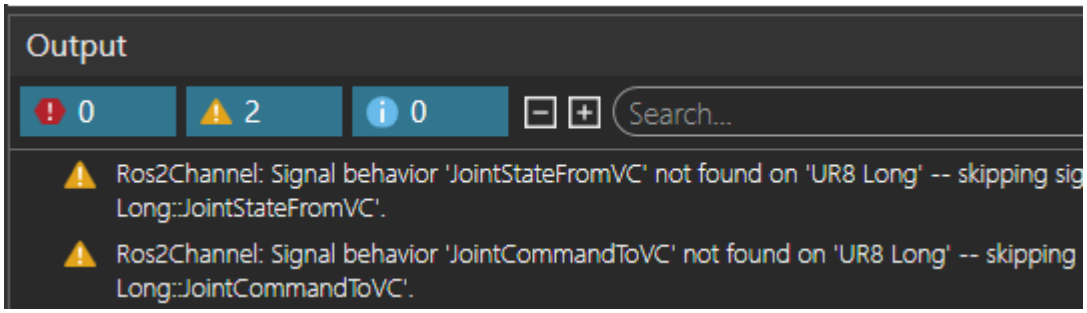
A12. Check the configuration and signal pairings, then reload the layout:

First check the ROS2 Pixi paths: the bridge cannot start if **runtime.py** cannot find **local_setup.bat**, **pixi.toml**, or **.pixi\envs\default\Library\bin**. Confirm the installation is under **C:\pixi_ws** or, for the supported legacy Lyrical layout, **D:\pixi_ws\ros2-windows**. Otherwise, update **runtime.py** or set the **VC_ROS2_PIXI_WS** and **VC_ROS2_INSTALL** environment variables.

1. Check the VC Output panel for signal-pairing warnings or errors.
2. Confirm the XML is loaded through ROS2 Config. Each referenced DataPacket behaviour must exist on the target component with exactly the XML signal name, including capitalisation. Verify the runtime or bridgeLauncher path.
3. Save the layout, clear it from VC, and reload it. Do not use Clear in ROS2 Config—it deletes the stored XML.
4. Start the simulation. If the bridge still fails to launch, check the Output panel and bridge window for further warnings or errors.

Example Output panel warnings and errors



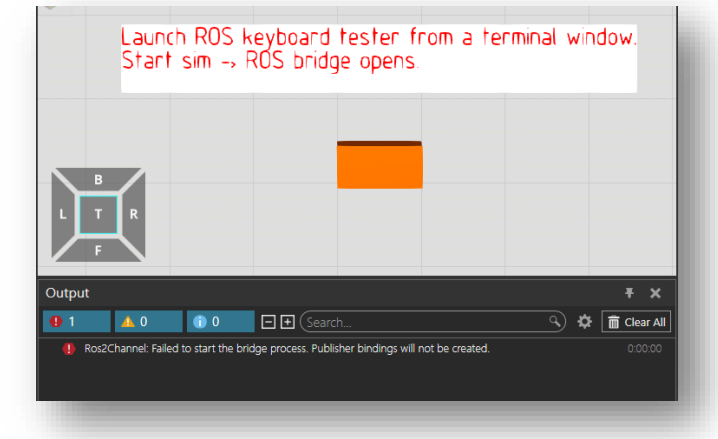


Missing JointStateFromVC and JointCommandToVC behaviours on UR8 Long: the corresponding mappings were skipped.

```
Ros2Channel: Failed to create publisher for topic
Ros2Channel: Failed to create subscriber for topic
```

Publisher/subscriber creation errors do not identify the cause. Follow the checks above, then save, clear, reload, and restart the simulation.

Q13. VC reports “Failed to start the bridge process. Publisher bindings will not be created.” What should I do?

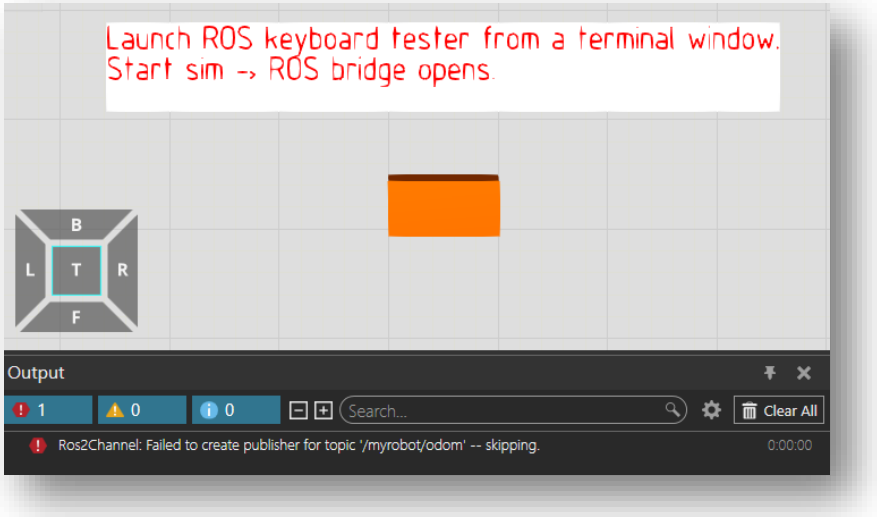


A13. VC could not run the command in the XML **bridgeLauncher** attribute. Complete these checks in order:

1. Confirm that **run_bridge.py** exists in the VCROS2Connection folder.
2. Confirm that **bridgeLauncher** uses the correct folder and either **--distro jazzy** or **--distro lyrical**.
3. Confirm that Python is available from Command Prompt and that the package path contains no spaces.
4. Confirm that the ROS2 Pixi paths required by **runtime.py** are valid.

5. In **ROS2 Config**, select **Browse...**, choose the corrected XML file, select **Upload**, and review the **Status** field.
6. Start the simulation again. Do not continue until the bridge window opens and **/vcbridge** appears in **ros2 node list**.

Q14. VC reports "Failed to create publisher" or "Failed to create subscriber." What should I do?



A14. The bridge started, but VC could not create one or more topic mappings. Complete these steps:

1. Check the VC Output panel for the affected topic and any missing DataPacket behaviour or signal.
2. Confirm that the XML topic, message type, component, and signal names match the layout exactly, including capitalisation.
3. Reset the simulation.
4. Save the layout, close or clear the layout from VC, and load it again. Do not select **Clear** in ROS2 Config because that deletes the stored XML configuration.
5. Start the simulation and check the Output panel again. For the demo layout, no publisher or subscriber creation errors should appear.

Appendix A: XML Pairing

The Ros2Connectivity XML configures how VC signals are connected to ROS2 topics. Store the XML content in the ConfigurationXML string property on the persistent layout item Ros2ConnectivityConfiguration. The XML content is read each time the simulation is opened.

Use this appendix when creating or reviewing the XML pairing file. It highlights the required XML structure, connector direction, key attributes, and common configuration errors.

Full example

```
<Ros2Connectivity bridgeLauncher "python C:\\VCROS2Connection\\run_bridge.py --distro
lyrical"

bridgeStartTimeoutSeconds="60">
  <Subscribers>
    <Subscriber signal="MobileRobot::VelocityIn" topic="/myrobot/cmd_vel"
ros2Type="geometry_msgs/msg/Twist" packetTypeId="vc/motion/twist"
qosReliability="reliable" qosDurability="volatile" qosDepth="10" />
  </Subscribers>
  <Publishers>
    <Publisher signal="MobileRobot::ImuOut" topic="/myrobot/imu"
ros2Type="sensor_msgs/msg/Imu" qosReliability="reliable" qosDurability="volatile"
qosDepth="10">
      <Constant name="frame_id" type="string" value="base_link" />
    </Publisher>
    <Publisher signal="MobileRobot::BatteryOut" topic="/myrobot/battery_state"
ros2Type="sensor_msgs/msg/BatteryState">
      <Constant name="temperature" type="float64" value="22.5" />
      <Constant name="power_supply_status" type="int32" value="1" />
      <Constant name="present" type="bool" value="true" />
    </Publisher>
  </Publishers>
</Ros2Connectivity>
```

Elements and attributes

Item	Required	Meaning
Ros2Connectivity	Yes	Root element for the ROS2 connectivity configuration.
bridgeLauncher	Yes	Windows command used to start the selected bridge executable. Use a path without spaces, for example python C:\VCROS2Connection\run_bridge.py --distro lyrical.

bridgeStartTimeoutSeconds	No	Non-negative wait time for bridge startup. Invalid or negative values use the implementation default.
Subscribers	No	Container for zero or more ROS2-to-VC subscriber bindings.
Subscriber	Per binding	Receives ROS2 messages and writes them to a VC DataPacket signal.
Publishers	No	Container for zero or more VC-to-ROS2 publisher bindings.
Publisher	Per binding	Sends VC DataPacket signal values to a ROS2 topic.
signal	Yes	VC signal address in component::signal format, for example MobileRobot::VelocityIn. The target behavior must be a DataPacket signal.
topic	Yes	ROS2 topic name. Reusing a topic produces a parser warning.
ros2Type	Yes	Supported ROS2 message type, for example geometry_msgs/msg/Twist or sensor_msgs/msg/Imu.
packetTypeId	Subscriber only, conditional	Type of packet written to a VC signal. Can be omitted when the target signal has a fixed PacketTypeId. Required when the target signal accepts any packet type. Replaces the legacy vcType attribute.
qosReliability	No	reliable or best_effort. Invalid values use the default.
qosDurability	No	volatile or transient_local. Invalid values use the default.
qosDepth	No	Positive integer queue depth. Invalid values use the default.
Constant	Publisher child, no	Adds a fixed compatible ROS2 field after VC packet conversion. It can complete required ROS2 fields or override a converted field with the same name.
Constant name	Yes	ROS2 message field name or dotted path, for example frame_id or header.frame_id.
Constant type	Yes	Supported constant value type: string, float64, int32, or bool.
Constant value	Yes	Field value. Boolean values accept true, false, 1, or 0.

Direction and packet types

- A Publisher sends VC signal data to ROS2.
- A Subscriber receives a ROS2 topic and writes a DataPacket to a VC signal.
- Use packetTypeId instead of the legacy vcType attribute.
- For wildcard subscriber signals, packetTypeId is required.
- Use Constant values to provide required ROS2 fields, such as frame_id.

Configuration errors

- Missing bridgeLauncher, signal, topic, or ros2Type makes the related configuration invalid.
- Malformed XML reports line and column information.
- Unknown wrapper or container elements are accepted, but only recognized binding elements create mappings.
- Constant entries outside Publisher are ignored.

Appendix B: DataPackets

VC 5.2 includes six built-in DataPacket types that are relevant for ROS2 connectivity and mobile-robot examples.

Use this appendix to choose the correct DataPacket type for VC signals. It includes both built-in packet types and guidance for custom dynamic packets.

Packet	Packet type ID	Fields

Twist	vc/motion/twist	linear, angular
Velocity command	vc/motion/velocity-command	linear, angular
Odometry	vc/navigation/odometry	time, pose_position, pose_orientation, twist_linear, twist_angular
IMU	vc/sensor/imu	time, orientation, angular_velocity, linear_acceleration
Joint state	vc/sensor/joint-state	time, name, position, velocity, effort
Laser scan	vc/sensor/laser-scan	time, angle_min, angle_max, angle_increment, ranges

Dynamic custom packets

V5.2 supports custom DataPackets with user-defined type names. Use the required Custom: prefix and a stable name, for example Custom:mission-status.

Custom packets start empty and act as dynamic key-value packets. Add field names and values, and check the packet type ID before reading custom fields.

With ROS2, matching packet fields map to matching ROS2 fields. Nested names use underscores, such as pose_position. Unmatched fields are skipped when publishing, and subscribed ROS2 fields are added to the custom packet.

Use `Constant` values for required fields such as `frame_id`.

Appendix C: Supported ROS2 Message Types

This appendix lists the ROS2 message types currently supported by VCROS2Connection. Use these values in the ros2Type attribute of the XML pairing file when configuring publishers and subscribers. The tables show which VC fields can be mapped to each ROS2 message type. If a required ROS2 field is not available from the VC DataPacket, add a Constant entry in the XML configuration, for example to provide a fixed frame_id.

std_msgs

ROS2 message type	VC fields
std_msgs/msg/Bool	value (Bool)
std_msgs/msg/Empty	No fields; trigger signal
std_msgs/msg/Float32	value (Real)

std_msgs/msg/Float64	value (Real)
std_msgs/msg/Int32	value (Int)
std_msgs/msg/Int64	value (Int)
std_msgs/msg/String	value (String)
std_msgs/msg/UInt8	value (Int)
std_msgs/msg/UInt32	value (Int)

geometry_msgs

ROS2 message type	VC fields
geometry_msgs/msg/Point	x, y, z (Real)
geometry_msgs/msg/Pose	position (Vector), orientation (Vector/Quaternion)
geometry_msgs/msg/PoseStamped	time, frame_id, pose_position, pose_orientation
geometry_msgs/msg/Quaternion	x, y, z, w (Real)
geometry_msgs/msg/TransformStamped	time, frame_id, child_frame_id, transform_translation, transform_rotation
geometry_msgs/msg/Twist	linear (Vector), angular (Vector)
geometry_msgs/msg/TwistStamped	time, frame_id, twist_linear, twist_angular
geometry_msgs/msg/Vector3	x, y, z (Real)
geometry_msgs/msg/Wrench	force (Vector), torque (Vector)
geometry_msgs/msg/WrenchStamped	time, frame_id, wrench_force, wrench_torque

nav_msgs

ROS2 message type	VC fields
nav_msgs/msg/Odometry	time, frame_id, child_frame_id, pose_position, pose_orientation, twist_linear, twist_angular

sensor_msgs

ROS2 message type	VC fields
sensor_msgs/msg/BatteryState	time, voltage, temperature, current, charge, capacity, design_capacity, percentage, power_supply_status, power_supply_health, power_supply_technology, present, cell_voltage, cell_temperature, location, serial_number

sensor_msgs/msg/Imu	time, frame_id, orientation, angular_velocity, linear_acceleration
sensor_msgs/msg/JointState	time, name, position, velocity, effort
sensor_msgs/msg/Range	time, frame_id, radiation_type, field_of_view, min_range, max_range, range
sensor_msgs/msg/LaserScan	time, frame_id, angle_min, angle_max, angle_increment, time_increment, scan_time, range_min, range_max, ranges, intensities
sensor_msgs/msg/Temperature	time, frame_id, temperature, variance

trajectory_msgs

ROS2 message type	VC fields
trajectory_msgs/msg/JointTrajectoryPoint	positions, velocities, accelerations, effort, time_from_start