You're reading the documentation for an older, but still supported, version of ROS 2. For information on the latest version, please have a look at Iron.

Using event handlers

Goal: Learn about event handlers in ROS 2 launch files

Tutorial level: Intermediate

Time: 15 minutes

Table of Contents

Background

Launch in ROS 2 is a system that executes and manages user-defined processes. It is responsible for monitoring the state of processes it launched, as well as reporting and reacting to changes in the state of those processes. These changes are called events and can be handled by registering an event handler with the launch system. Event handlers can be registered for specific events and can be useful for monitoring the state of processes. Additionally, they can be used to define a complex set of rules which can be used to dynamically modify the launch file.

This tutorial shows usage examples of event handlers in ROS 2 launch files.

Prerequisites

This tutorial uses the turtlesim package. This tutorial also assumes you have created a new package of build type `ament_python` called `launch_tutorial`.

This tutorial extends the code shown in the Using substitutions in launch files tutorial.

Using event handlers

1 Event handlers example launch file

Create a new file called `example_event_handlers.launch.py` file in the `launch` folder of the `launch_tutorial` package.

```python
from launch_ros.actions import Node

from launch import LaunchDescription
from launch.actions import (DeclareLaunchArgument, EmitEvent, ExecuteProcess,
                            LogInfo, RegisterEventHandler, TimerAction)
from launch.conditions import IfCondition
from launch.event_handlers import (OnExecutionComplete, OnProcessExit,
                                OnProcessIO, OnProcessStart, OnShutdown)
from launch.events import Shutdown
from launch.substitutions import (EnvironmentVariable, FindExecutable,
                                LaunchConfiguration, LocalSubstitution,
                                PythonExpression)


def generate_launch_description():
    turtlesim_ns = LaunchConfiguration('turtlesim_ns')
    use_provided_red = LaunchConfiguration('use_provided_red')
    new_background_r = LaunchConfiguration('new_background_r')

    turtlesim_ns_launch_arg = DeclareLaunchArgument(
        'turtlesim_ns',
        default_value='turtlesim1'
    )
    use_provided_red_launch_arg = DeclareLaunchArgument(
        'use_provided_red',
        default_value='False'
    )
    new_background_r_launch_arg = DeclareLaunchArgument(
        'new_background_r',
        default_value='200'
    )

    turtlesim_node = Node(
        package='turtlesim',
        namespace=turtlesim_ns,
        executable='turtlesim_node',
        name='sim'
    )
    spawn_turtle = ExecuteProcess(
        cmd=[[
            FindExecutable(name='ros2'),
            ' service call ',
            turtlesim_ns,
            '/spawn ',
            'turtlesim/srv/Spawn ',
            '"{x: 2, y: 2, theta: 0.2}"'
        ]],
        shell=True
    )
    change_background_r = ExecuteProcess(
        cmd=[[
            FindExecutable(name='ros2'),
            ' param set ',
            turtlesim_ns,
            '/sim background_r ',
            '120'
        ]],
        shell=True
    )
    change_background_r_conditioned = ExecuteProcess(
        condition=IfCondition(
            PythonExpression([
                new_background_r,
                ' == 200',
                ' and ',
                use_provided_red
            ])
        ),
        cmd=[[
            FindExecutable(name='ros2'),
            ' param set ',
            turtlesim_ns,
            '/sim background_r ',
            new_background_r
        ]],
        shell=True
    )

    return LaunchDescription([
        turtlesim_ns_launch_arg,
        use_provided_red_launch_arg,
        new_background_r_launch_arg,
        turtlesim_node,
        RegisterEventHandler(
            OnProcessStart(
                target_action=turtlesim_node,
                on_start=[
                    LogInfo(msg='Turtlesim started, spawning turtle'),
                    spawn_turtle
                ]
            )
        ),
        RegisterEventHandler(
            OnProcessIO(
                target_action=spawn_turtle,
                on_stdout=lambda event: LogInfo(
                    msg='Spawn request says "{}"'.format(
                        event.text.decode().strip())
                )
            )
        ),
        RegisterEventHandler(
            OnExecutionComplete(
                target_action=spawn_turtle,
                on_completion=[
                    LogInfo(msg='Spawn finished'),
                    change_background_r,
                    TimerAction(
                        period=2.0,
                        actions=[change_background_r_conditioned],
                    )
                ]
            )
        ),
        RegisterEventHandler(
            OnProcessExit(
                target_action=turtlesim_node,
                on_exit=[
                    LogInfo(msg=(EnvironmentVariable(name='USER'),
                         ' closed the turtlesim window')),
                    EmitEvent(event=Shutdown(
                        reason='Window closed'))
                ]
            )
        ),
        RegisterEventHandler(
            OnShutdown(
                on_shutdown=[LogInfo(
                    msg=['Launch was asked to shutdown: ',
                        LocalSubstitution('event.reason')]
                )]
            )
        ),
    ])
```

`RegisterEventHandler` actions for the `OnProcessStart`, `OnProcessIO`, `OnExecutionComplete`, `OnProcessExit`, and `OnShutdown` events were defined in the launch description.

The `OnProcessStart` event handler is used to register a callback function that is executed when the turtlesim node starts. It logs a message to the console and executes the `spawn_turtle` action when the turtlesim node starts.

```
RegisterEventHandler(
    OnProcessStart(
        target_action=turtlesim_node,
        on_start=[
            LogInfo(msg='Turtlesim started, spawning turtle'),
            spawn_turtle
        ]
    )
),
```

The `OnProcessIO` event handler is used to register a callback function that is executed when the `spawn_turtle` action writes to its standard output. It logs the result of the spawn request.

```
RegisterEventHandler(
    OnProcessIO(
        target_action=spawn_turtle,
        on_stdout=lambda event: LogInfo(
            msg='Spawn request says "{}"'.format(
                event.text.decode().strip())
        )
    )
),
```

The `OnExecutionComplete` event handler is used to register a callback function that is executed when the `spawn_turtle` action completes. It logs a message to the console and executes the `change_background_r` and `change_background_r_conditioned` actions when the spawn action completes.

```
RegisterEventHandler(
    OnExecutionComplete(
        target_action=spawn_turtle,
        on_completion=[
            LogInfo(msg='Spawn finished'),
            change_background_r,
            TimerAction(
                period=2.0,
                actions=[change_background_r_conditioned],
            )
        ]
    )
),
```

The `OnProcessExit` event handler is used to register a callback function that is executed when the turtlesim node exits. It logs a message to the console and executes the `EmitEvent` action to emit a `Shutdown` event when the turtlesim node exits. It means that the launch process will shutdown when the turtlesim window is closed.

```
RegisterEventHandler(
    OnProcessExit(
        target_action=turtlesim_node,
        on_exit=[
            LogInfo(msg=(EnvironmentVariable(name='USER'),
                    ' closed the turtlesim window')),
            EmitEvent(event=Shutdown(
                reason='Window closed'))
        ]
    )
),
```

Finally, the `OnShutdown` event handler is used to register a callback function that is executed when the launch file is asked to shutdown. It logs a message to the console why the launch file is asked to shutdown. It logs the message with a reason for shutdown like the closure of turtlesim window or `ctrl-c` signal made by the user.

```
RegisterEventHandler(
    OnShutdown(
        on_shutdown=[LogInfo(
            msg=['Launch was asked to shutdown: ',
                LocalSubstitution('event.reason')]
        )]
    )
),
```

Build the package

Go to the root of the workspace, and build the package:

```
colcon build
```

Also remember to source the workspace after building.

Launching example

Now you can launch the `example_event_handlers.launch.py` file using the `ros2 launch` command.

```
ros2 launch launch_tutorial example_event_handlers.launch.py turtlesim_ns:='turtlesim3'
use_provided_red:='True' new_background_r:=200
```

This will do the following:

1. Start a turtlesim node with a blue background
2. Spawn the second turtle
3. Change the color to purple
4. Change the color to pink after two seconds if the provided `background_r` argument is `200` and `use_provided_red` argument is `True`
5. Shutdown the launch file when the turtlesim window is closed

Additionally, it will log messages to the console when:

1. The turtlesim node starts
2. The spawn action is executed
3. The `change_background_r` action is executed
4. The `change_background_r_conditioned` action is executed
5. The turtlesim node exits

6. The launch process is asked to shutdown.

Documentation

The launch documentation provides detailed information about available event handlers.

Summary

In this tutorial, you learned about using event handlers in launch files. You learned about their syntax and usage examples to define a complex set of rules to dynamically modify launch files.