ROS 2 Launch Files - Complete Training Document

1. Introduction to Launch Files

What Are Launch Files?

Launch files are configuration scripts that automate starting multiple ROS 2 nodes with specific settings.

Before vs After Launch Files

🔴 BEFORE (Manual Startup):

```
bash
# Terminal 1:
ros2 run simple_service true_calculator_server

# Terminal 2:
ros2 run simple_service true_calculator_client 15 3 add

# Terminal 3:

ros2 run my_py_pkg simple_processor
```

🟢 AFTER (Launch Files):

```
bash
# ONE TERMINAL:

ros2 launch simple_service complete_system.launch.py
```

Benefits Comparison

Without Launch Files	With Launch Files

Manual node starting	One-command startup
Inconsistent configurations	Reproducible setups
Hard to share setups	Easy team collaboration
Error-prone manual steps	Automated, reliable

Where We Use Launch Files

- Robot startups - Sensors, control, navigation all at once
- Testing setups - Consistent test environments
- Simulations - Multiple nodes together
- Production systems - Reliable deployments

---

2. Basic Launch File Structure

📁 Project Structure

text

```
ros2_ws1/
└── src/
    ├── my_py_pkg/          #  Your first package
    │   ├── package.xml
    │   ├── setup.py
    │   ├── launch/
    │   └── my_py_pkg/
    └── simple_service/     #  Your second package
        ├── package.xml
        ├── setup.py
        ├── launch/

        └── simple_service/
```

Basic Launch File Template

File: `simple_service/launch/basic_calculator.launch.py`

python

```
#!/usr/bin/env python3
from launch import LaunchDescription
from launch_ros.actions import Node

def generate_launch_description():
    """
    🎯 Basic launch file for calculator service
    """

    # Create launch description
    ld = LaunchDescription()

    # Calculator server node
    calculator_server = Node(
        package='simple_service',    # 📦 Package name
        executable='true_server',  # 🚀 Executable name
        name='math_server',         # 🏷️ Custom node name
        output='screen'             # 📺 Show output
    )

    # Calculator client node
    calculator_client = Node(
        package='simple_service',
        executable='true_client',
        name='math_client',
        arguments=['15', '3', 'add'],  # 🔧 Fixed calculation
        output='screen'
    )

    # Add nodes to launch description
    ld.add_action(calculator_server)
    ld.add_action(calculator_client)

    return ld
```

Package Setup Requirements

Update `simple_service/package.xml`:

xml

```xml
<exec_depend>launch</exec_depend>

<exec_depend>launch_ros</exec_depend>
```

Update `simple_service/setup.py`:

python

```python
from setuptools import setup
import os
from glob import glob

package_name = 'simple_service'

setup(
    name=package_name,
    version='0.0.0',
    packages=[package_name],
    data_files=[
        ('share/ament_index/resource_index/packages',
            ['resource/' + package_name]),
        ('share/' + package_name, ['package.xml']),
        # 🆕 ADD LAUNCH FILES
        (os.path.join('share', package_name, 'launch'),
         glob('launch/*.launch.py')),
    ],
    install_requires=['setuptools'],
    zip_safe=True,
    entry_points={
        'console_scripts': [
            'true_calculator_server = simple_service.true_server:main',
            'true_calculator_client = simple_service.true_client:main',
        ],
    },

)
```

Usage Commands

```
bash
```

```bash
# Build your package
cd ~/ros2_ws1
colcon build --packages-select simple_service

# Source your workspace
source install/setup.bash

# Use launch file instead of manual commands
```

```
ros2 launch simple_service basic_calculator.launch.py
```

3. Substitutions & Dynamic Launch Files

Substitution Types

Substitution	Purpose	Example
`LaunchConfiguration`	Get launch arguments	`LaunchConfiguration('a')`
`PathJoinSubstitution`	Build file paths	`PathJoinSubstitution([pkg_path, 'config'])`

📝 Dynamic Calculator Example

File: `simple_service/launch/dynamic_calculator.launch.py`

```
python
```

```python
#!/usr/bin/env python3
from launch import LaunchDescription
from launch_ros.actions import Node
from launch.actions import DeclareLaunchArgument, TimerAction
from launch.substitutions import LaunchConfiguration
```

```python
def generate_launch_description():
    ld = LaunchDescription()

    # 🎯 DEFINE LAUNCH ARGUMENTS
    a_arg = DeclareLaunchArgument(
        'a',
        default_value='10',
        description='First number for calculation'
    )

    b_arg = DeclareLaunchArgument(
        'b',
        default_value='5',
        description='Second number for calculation'
    )

    operation_arg = DeclareLaunchArgument(
        'operation',
        default_value='add',
        description='Math operation: add, subtract, multiply, divide'
    )

    delay_arg = DeclareLaunchArgument(
        'client_delay',
        default_value='3.0',
        description='Delay before starting client (seconds)'
    )

    # 🎯 SERVER
    calculator_server = Node(
        package='simple_service',
        executable='true_server',
        name='math_server',
        output='screen'
    )

    # 🎯 DYNAMIC CLIENT
    calculator_client = Node(
        package='simple_service',
        executable='true_client',
        name='math_client',
        arguments=[
            LaunchConfiguration('a'),        # 🎯 Dynamic number
            LaunchConfiguration('b'),        # 🎯 Dynamic number
            LaunchConfiguration('operation') # 🎯 Dynamic operation
        ],
        output='screen'
    )

    # 🎯 Add components
    ld.add_action(a_arg)
    ld.add_action(b_arg)
    ld.add_action(operation_arg)
    ld.add_action(delay_arg)
    ld.add_action(calculator_server)

    # 🎯 Add client with dynamic delay
    ld.add_action(TimerAction(
        period=LaunchConfiguration('client_delay'),
        actions=[calculator_client]
    ))
```

```
    return ld
```

🖥️ Dynamic Usage Commands

```bash
# Different calculations with same launch file:

# Default values (10 + 5)
ros2 launch simple_service dynamic_calculator.launch.py

# Custom addition (25 + 15)
ros2 launch simple_service dynamic_calculator.launch.py a:=25 b:=15 operation:=add

# Multiplication with shorter delay
ros2 launch simple_service dynamic_calculator.launch.py a:=8 b:=7 operation:=multiply client_delay:=2.0

# Show available arguments
ros2 launch simple_service dynamic_calculator.launch.py --show-args
```

📁 YAML Configuration Files

File: `simple_service/config/calculator_config.yaml`

yaml

```yaml
math_server:
  ros__parameters:
    #  LOGGING CONFIGURATION
    log_level: "INFO"

    # ⚡ PERFORMANCE SETTINGS
    timeout_seconds: 30.0

    #  CALCULATION SETTINGS
    allow_division_by_zero: false
```

Using YAML in Launch:

python

```python
from launch.substitutions import PathJoinSubstitution
from launch_ros.substitutions import FindPackageShare

config_path = PathJoinSubstitution([
    FindPackageShare('simple_service'),
    'config',
    'calculator_config.yaml'
])

server = Node(
    package='simple_service',
    executable='true_server',
    parameters=[config_path]  # 🎯 Load from YAML
)
```

4. Event Handlers

Event Types

Event Handler	Triggers When	Use Case
`OnProcessStart`	Process starts	Start dependent nodes
`OnProcessExit`	Process finishes/crashes	Error recovery

📝 Reactive Calculator Example

File: `simple_service/launch/reactive_calculator.launch.py`

python

```python
#!/usr/bin/env python3
from launch import LaunchDescription
from launch_ros.actions import Node
from launch.actions import RegisterEventHandler, LogInfo, TimerAction
from launch.event_handlers import OnProcessStart, OnProcessExit

def generate_launch_description():
    ld = LaunchDescription()

    # 🎯 CORE NODES
    calculator_server = Node(
        package='simple_service',
        executable='true_server',
        name='math_server',
        output='screen'
    )

    calculator_client = Node(
        package='simple_service',
        executable='_client',
        name='math_client',
        arguments=['15', '3', 'add'],
        output='screen'
    )
```

```
# 🎯 EVENT 1: When server starts
server_ready_handler = RegisterEventHandler(
    OnProcessStart(
        target_action=calculator_server,
        on_start=[
            LogInfo(msg='✅ Calculator server is online!'),
            LogInfo(msg='🚀 Starting client in 3 seconds...'),
            TimerAction(
                period=3.0,
                actions=[calculator_client]
            )
        ]
    )
)

# 🎯 EVENT 2: When calculation completes
calculation_complete_handler = RegisterEventHandler(
    OnProcessExit(
        target_action=calculator_client,
        on_exit=[
            LogInfo(msg='🧮 Calculation completed!'),
            LogInfo(msg='📊 Result: 15 + 3 = 18')
        ]
    )
)

# 🎯 EVENT 3: If server crashes
server_fail_handler = RegisterEventHandler(
    OnProcessExit(
        target_action=calculator_server,
        on_exit=[
            LogInfo(msg='⚠️ Server crashed! Emergency shutdown...')
        ]
    )
)

ld.add_action(calculator_server)
ld.add_action(server_ready_handler)
ld.add_action(calculation_complete_handler)
ld.add_action(server_fail_handler)
```

```
return ld
```

🖥️ Event Testing Commands

```bash
# Test reactive launch file
ros2 launch simple_service reactive_calculator.launch.py

# Test error scenarios (kill server to see events trigger)
```

5. Running Nodes from Different Packages in One Launch File

Concept:

You can start nodes from multiple packages in your workspace in a single launch file.

📝 Multi-Package System Example

File: `simple_service/launch/multi_package_system.launch.py`

```python
#!/usr/bin/env python3
from launch import LaunchDescription
from launch_ros.actions import Node

def generate_launch_description():
    ld = LaunchDescription()

    # 🎯 FROM simple_service PACKAGE: Calculator nodes
    calculator_server = Node(
        package='simple_service',
        executable='true_calculator_server',
        name='calculator_server',
        output='screen'
    )

    calculator_client = Node(
    package='simple_service',
    executable='true_calculator_client',
    name='calculator_client',
    arguments=['25', '5', 'multiply'],
    output='screen'
)

# 🎯 FROM my_py_pkg PACKAGE: Custom processor node
# (Assuming you have a node called 'simple_processor')
data_processor = Node(
    package='my_py_pkg',
    executable='simple_processor',
    name='data_processor',
    output='screen'
)

# 🎯 FROM demo_nodes_cpp PACKAGE: Standard ROS 2 nodes
demo_talker = Node(
    package='demo_nodes_cpp',
    executable='talker',
    name='demo_talker',
    remappings=[
        ('chatter', 'input_data')  # 🔄 Custom topic name
    ],
    output='screen'
)

demo_listener = Node(
    package='demo_nodes_cpp',
    executable='listener',
    name='demo_listener',
    remappings=[
        ('chatter', 'processed_data')  # 🔄 Listen to processed data
    ],
    output='screen'
)

# Add all nodes from different packages
ld.add_action(calculator_server)
ld.add_action(calculator_client)
ld.add_action(data_processor)
ld.add_action(demo_talker)
ld.add_action(demo_listener)
```

```python
return ld
```

📝 Simple Processor Node for my_py_pkg

File: `my_py_pkg/my_py_pkg/simple_processor.py`

python

```python
#!/usr/bin/env python3
import rclpy
from rclpy.node import Node
from std_msgs.msg import String

class SimpleProcessor(Node):
    def __init__(self):
        super().__init__('simple_processor')

        # Create publisher
        self.publisher = self.create_publisher(String, 'processed_data',
10)

        # Create subscription
        self.subscription = self.create_subscription(
            String,
            'input_data',
            self.data_callback,
            10
        )

        self.get_logger().info('🔄 Simple Processor node started!')

    def data_callback(self, msg):
        # Process incoming data
        processed_msg = String()
        processed_msg.data = f"PROCESSED: {msg.data}"
        self.publisher.publish(processed_msg)
        self.get_logger().info(f'📊 Processed: {msg.data}')

def main():
    rclpy.init()
    node = SimpleProcessor()
    rclpy.spin(node)
    node.destroy_node()
    rclpy.shutdown()

if __name__ == '__main__':
    main()
```

Update `my_py_pkg/setup.py`:

python

```
from setuptools import setup
import os
from glob import glob

package_name = 'my_py_pkg'

setup(
    name=package_name,
    version='0.0.0',
    packages=[package_name],
    data_files=[
        ('share/ament_index/resource_index/packages',
            ['resource/' + package_name]),
        ('share/' + package_name, ['package.xml']),
    ],
    install_requires=['setuptools'],
    zip_safe=True,
    entry_points={
        'console_scripts': [
            'simple_processor = my_py_pkg.simple_processor:main',
        ],
    },
)
```

🖥️ Multi-Package Usage Commands

bash

```
# Navigate to your workspace
cd ~/ros2_ws1

# Build BOTH your packages
colcon build --packages-select my_py_pkg simple_service

# Source your workspace
source install/setup.bash

# 🎯 Start ALL nodes from different packages with ONE command:
ros2 launch simple_service multi_package_system.launch.py
```

🔍 Expected Output:

```text
[calculator_server] 🎯 True Calculator Server ready!
[calculator_client] 📤 Sent request: 25 multiply 5
[calculator_server] 📥 Received: 25 multiply 5
[calculator_client] ✅ Multiplication successful
[calculator_client] 🎯 Result: 125

[data_processor] 🔄 Simple Processor node started!
[demo_talker] Publishing: "Hello World: 0"
[data_processor] 📊 Processed: Hello World: 0

[demo_listener] I heard: "PROCESSED: Hello World: 0"
```

🚀 Summary & Best Practices

✅ What You Learned

- 🎯 Basic launch files - Start multiple nodes together
- 🔄 Dynamic launch files - Substitutions and arguments
- 🎯 Event handling - Reactive startup sequences
- 📦 Multi-package systems - Unified workspace management