FeatX Technical Guideline Documentation

Jude Gyimah[1], Henriette Knopp[1], Thorsten Berger[1] and Patrizio Pelliccione[2]

This is the official guideline documentation for the featX tool. Its intended to assist users in the setup, installation, modelling features, binding features, implementation and control of features, as well as general information about the tool.

Contents

I. Prerequisites and Dependencies

(I) Ubuntu 22.04.4 LTS or higher
(II) ROS2 Iron or higher
(III) Python3
(IV) C++ 11 or higher
(V) Pytest

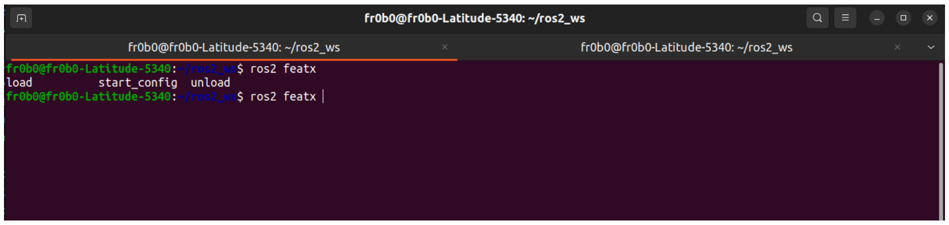
II. Components of FeatX Suite

1) **featxcli:** CLI, DSL Semantic Implementation, Configurator
2) **featxbinder:** Interface for static compile-time (Early) binding and rclcpp plugins (Dynamic)
3) **featx_interfaces:** Service channels for rclpy plugin implementation and its load and unload operations

III. Setup and Installation

1) Clone FeatX Suite from here into ros2 workspace.
2) Build work space (colcon build)
3) Check the inclusion of "featx"command and command verbs: ros2 featx [tab] [tab]. See Fig. 1

[1] Faculty of Computer Science, Chair of Software Engineering, Ruhr University, Bochum, Germany `thorsten.berger@rub.de`, `jude.gyimah@rub.de`, `henriette.knopp@rub.de`
[2] Gran Sasso Science Institute (GSSI), L'Aquila, Italy `patrizio.pelliccione@gssi.it`


fr0b0@fr0b0-Latitude-5340: ~/ros2_ws
fr0b0@fr0b0-Latitude-5340: ~/ros2_ws
fr0b0@fr0b0-Latitude-5340: ~/ros2_ws

```
fr0b0@fr0b0-Latitude-5340:~/ros2_ws$ ros2 featx
load          start_config  unload
fr0b0@fr0b0-Latitude-5340:~/ros2_ws$ ros2 featx
```

Fig. 1: Featx command verb check

IV. The Configurator

The configurator can be found in **featxcli/featxcli/configurator.py**. The configurator checks the following rules:

1) Feature binding consistency: That the binding definitions specified align with the bindings configured.
2) A static child feature thats selected cannot depend on a dynamic parent.
3) An early bound child feature cannot depend on a late one.
4) Features that exclude each other cannot be selected simultaneously.
5) A selected feature cannot have a deselected parent feature.

For each attempt to load and unload a feature, the above rules are checked by the configurator. If conflicts are found, the configurator blocks the operation and prompts the user in the console. However if no conflicts are found, the operation is allowed to execute.

V. FeatX Domain-specific Language

A. Defining Features

To model feature definitions of a robot in featX, in the **featxcli/featxcli/model/** directory of the FeatX Suite, open the **features.json** file and complete the feature model template provided. To extend the **root** feature of feature model hierarchy, a **sub** key can be used to provide an array of sub-feature objects (See Listing 1).

```
{    "name":"root",
    "sub":[{
            "name":"attach_service",
            "isOptional": false,
            "includes":"move_shelf_to_ship",
            "excludes":"",
            "bindingTimeAllowed": "Early",
            "bindingModeAllowed": "Static",
            "sub":[
                {
                    "name":"approach_service_server",
                    "isOptional": false,
                    "includes":"",
                    "excludes":"",
                    "bindingTimeAllowed": "Early",
                    "bindingModeAllowed": "Dynamic"
                }]
        }]}
```

Listing 1: FeatX feature definition

B. Configuring Features

To configure feature definitions, in the **featxcli/featxcli/model/** directory of the FeatX Suite, open the **configs.json** file and complete the configuration model template provided. To extend the configurations, a feature **name** reference, isSelected, bindingTime and bindingMode , bundled as an objects can be added to the **configs** array (See Listing 2).

```
{"configs":[
        {
            "name": "approach_service_server",
            "isSelected": true,
            "bindingTime": "Early",
            "bindingMode": "Dynamic"
        },
        {
            "name": "nav2_amcl",
            "isSelected": true,
            "bindingTime": "Late",
            "bindingMode": "Static"
        },
        {
            "name": "move_shelf_to_ship",
            "isSelected": true,
            "bindingTime": "Late",
            "bindingMode": "Dynamic"
        }]}
```
Listing 2: FeatX configuration

VI. Implementing Feature Bindings

Feature must be implemented as ros2 nodes. In your ros2 workspace create a **/packages** directory where all features would be implemented in packages. The parent of the feature that holds the required functionality becomes the name of the package that houses that feature node. Features can be implemented as preprocessor bindings (static) and either rclcpp or rclpy plugins.

A. *Preprocessor bindings*

1) Create a c++ macros for the feature. For each macro, assign a **0** to signify that is not selected or a **1** to signify selection.
2) Create a c++ package and let the name of the package be the name of the parent feature that contains the configurable features.
3) Create features as nodes. For static c++ features, create them as manual ros2 compositions. Include the file **featxbinder/static_features.h** in the static node file.
4) Use conditional statements to perform conditional compiling of feature headers and add them to an executor. See Listing 3 as an example.

```cpp
//static_features.h
#define NAV2_MAP_SERVER 1
#define RTABMAP_ROS 0

//feature.h
#include "rclcpp/rclcpp.hpp"
#include "featxbinder/static_features.h"


#if NAV2_MAP_SERVER
  #include "mapping/nav2_map_server.hpp"
#else
  #pragma message("Static feature nav2_map_server not selected")
#endif

#if RTABMAP_ROS
  #include "mapping/rtabmap_ros.hpp"
#else
  #pragma message("Static feature rtabmap_ros not selected")
#endif


int main(int argc, char **argv){
    rclcpp::init(argc, argv);
    rclcpp::executors::SingleThreadedExecutor executor;

    #if RTABMAP_ROS
        auto rtabmap_ros_node = std::make_shared<RtabMapRos>();
        executor.add_node(rtabmap_ros_node);
    #endif

    #if NAV2_MAP_SERVER
        auto nav2_map_server_node = std::make_shared<Nav2MapServer>();
        executor.add_node(nav2_map_server_node);
    #endif

    executor.spin();

    rclcpp::shutdown();
    return 0;
}
```

Listing 3: Macros and conditional compiling

5) Add the static feature to **featxbinder/early.launch.py**

B. *rclcpp plugins*

1) Create a c++ package and let the name of the package be the name of the parent feature that contains the configurable features.
2) Create features as nodes. For dynamic(plugin) c++ features, create them as plugin compositions. Ensure that the class has the namespace **featx_plugin**.
3) Register the plugin and add the necessary configurations in the CMakeLists.txt to make them discoverable. For more information on how to do this, follow this link. See Listing 4 as an example.

```cpp
#include "rclcpp/rclcpp.hpp"

namespace featx_plugin{
    class CartographerSlam: public rclcpp::Node{
        public:
            CartographerSlam(const rclcpp::NodeOptions &options):Node("cartographer_slam", options){
                RCLCPP_INFO(this->get_logger(), "Hello cartographer_slam");
            }

    };
}

#include "rclcpp_components/register_node_macro.hpp"
RCLCPP_COMPONENTS_REGISTER_NODE(featx_plugin::CartographerSlam)
```

Listing 4: C++ plugin implementation

C. *rclpy plugins*

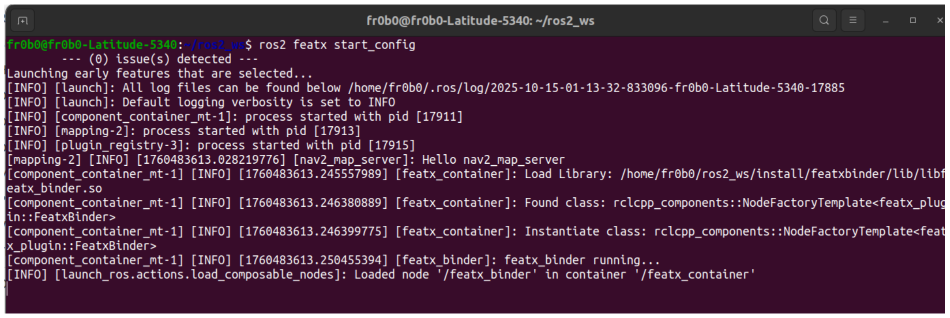
Since ros2 doesn't support rclpy plugins, we have extended the ros2 infrastructure with an importlib based plugin registry to accommodate this. Create an rclpy package and let the name of the package be the name of the parent feature that contains whatever configurable feature you want implement. Implement the feature as a ros2 node class. See See Listing 5 as an example.

```python
import rclpy
from rclpy.node import Node

class MoveShelfToShip(Node):
    def __init__(self):
        super().__init__("move_shelf_to_ship")
        self.get_logger().info("Hello move_shelf_to_ship")

def main(args=None):
    rclpy.init(args=args)
    node = MoveShelfToShip()
    rclpy.spin(node)
    rclpy.shutdown()

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

Listing 5: Python plugin implementation

VII. Starting a Configuration

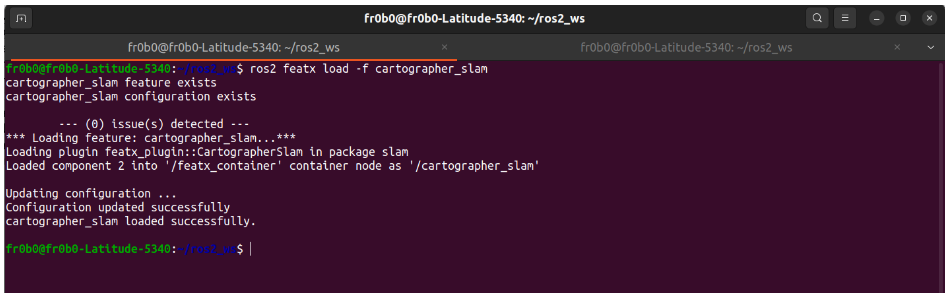
Command: ros2 featx start_config



```
fr0b0@fr0b0-Latitude-5340: ~/ros2_ws
fr0b0@fr0b0-Latitude-5340:~/ros2_ws$ ros2 featx start_config
        --- (0) issue(s) detected ---
Launching early features that are selected...
[INFO] [launch]: All log files can be found below /home/fr0b0/.ros/log/2025-10-15-01-13-32-833096-fr0b0-Latitude-5340-17885
[INFO] [launch]: Default logging verbosity is set to INFO
[INFO] [component_container_mt-1]: process started with pid [17911]
[INFO] [mapping-2]: process started with pid [17913]
[INFO] [plugin_registry-3]: process started with pid [17915]
[mapping-2] [INFO] [1760483613.028219776] [nav2_map_server]: Hello nav2_map_server
[component_container_mt-1] [INFO] [1760483613.245557989] [featx_container]: Load Library: /home/fr0b0/ros2_ws/install/featxbinder/lib/libf
eatx_binder.so
[component_container_mt-1] [INFO] [1760483613.246380889] [featx_container]: Found class: rclcpp_components::NodeFactoryTemplate<featx_plug
in::FeatxBinder>
[component_container_mt-1] [INFO] [1760483613.246399775] [featx_container]: Instantiate class: rclcpp_components::NodeFactoryTemplate<feat
x_plugin::FeatxBinder>
[component_container_mt-1] [INFO] [1760483613.250455394] [featx_binder]: featx_binder running...
[INFO] [launch_ros.actions.load_composable_nodes]: Loaded node '/featx_binder' in container '/featx_container'
```

Fig. 2: Starting a Configuration

The command above first launches all static features bound at compile time via the preprocessor embellished with conditional statements triggered via a launch file. It also launches the featxbinder container as well as the plugin registry. The configuration cannot be started if the DSL-based models have rule violations as shown in Fig. 2.



```
fr0b0@fr0b0-Latitude-5340:~/ros2_ws$ ros2 featx load -f cartographer_slam
cartographer_slam feature exists
cartographer_slam configuration exists

        --- (0) issue(s) detected ---
*** Loading feature: cartographer_slam...***
Loading plugin featx_plugin::CartographerSlam in package slam
Loaded component 2 into '/featx_container' container node as '/cartographer_slam'

Updating configuration ...
Configuration updated successfully
cartographer_slam loaded successfully.

fr0b0@fr0b0-Latitude-5340:~/ros2_ws$ |
```

Fig. 3: Loading a feature

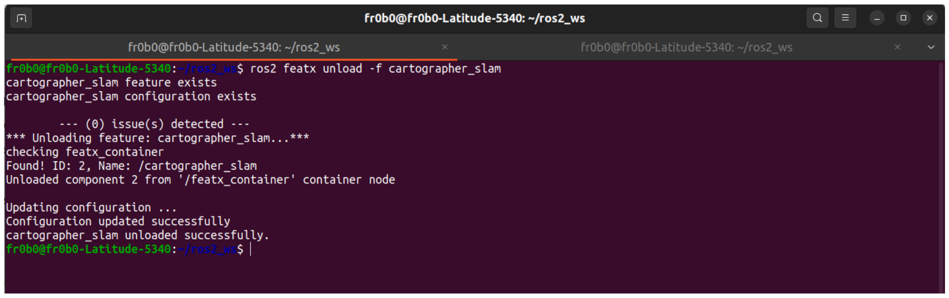
VIII. Loading a Feature

Command: ros2 featx load -f <feature_name>

This command checks the DSL models to ensure that the feature name specified exists, it checks the binding time and binding mode configuration to ensure that it supports runtime loading as well as the binding rules, then it creates a loadable plugin out of the corresponding feature implementation and starts the plugin node. The **-f** flag indicates that the string after the flag is the name of a feature specified in the DSL model. If the feature is loaded successfully or fails to be loaded, a console feedback message is displayed to that effect (see Fig. 3).

IX. Unloading a Feature

Command: ros2 featx unload -f <feature_name>



```
fr0b0@fr0b0-Latitude-5340:~/ros2_ws$ ros2 featx unload -f cartographer_slam
cartographer_slam feature exists
cartographer_slam configuration exists

        --- (0) issue(s) detected ---
*** Unloading feature: cartographer_slam...***
checking featx_container
Found! ID: 2, Name: /cartographer_slam
Unloaded component 2 from '/featx_container' container node

Updating configuration ...
Configuration updated successfully
cartographer_slam unloaded successfully.
fr0b0@fr0b0-Latitude-5340:~/ros2_ws$ |
```

Fig. 4: Unloading a feature

This command checks the DSL models to ensure that the feature name specified exists, it checks the binding time and binding mode configuration to ensure that it supports runtime unloading as well as fulfilling the binding rules, then it creates a unloadable plugin from memory and cleans up all traces of it. The **-f** flag indicates that the string after the flag is the name of a feature specified in the DSL model. If the feature is unloaded successfully or fails, a console feedback message is displayed to that effect (see Fig. 4).