crie o codigo e os comandos que tem que ser rodados, explicando cada parte de forma detalhada

Você vai criar 3 coisas principais: o pacote, o arquivo da mensagem `RoverEvents.msg`, configurar `CMakeLists.txt` e `package.xml`, e depois compilar e testar, sempre com comandos bem específicos.[1]

1. Criar o pacote CMake

No terminal (dentro do ambiente do curso), rode:

```
cd ~/ros2_ws/src
ros2 pkg create --build-type ament_cmake custom_interfaces
```

- `cd ~/ros2_ws/src`: entra na pasta `src` do seu workspace `ros2_ws`, que é onde ficam os pacotes.[1]
- `ros2 pkg create --build-type ament_cmake custom_interfaces`: cria um pacote novo chamado `custom_interfaces` usando CMake (`ament_cmake`), que é obrigatório para gerar mensagens customizadas em ROS2.[1]

2. Criar a pasta msg e o arquivo RoverEvents.msg

Agora entre na pasta do pacote e crie o arquivo da mensagem:

```
cd ~/ros2_ws/src/custom_interfaces
mkdir msg
touch msg/RoverEvents.msg
```

- `cd ~/ros2_ws/src/custom_interfaces`: entra na pasta do pacote recém-criado.[1]
- `mkdir msg`: cria a pasta `msg`, onde ROS2 espera encontrar definições de mensagens.[1]
- `touch msg/RoverEvents.msg`: cria o arquivo vazio da mensagem `RoverEvents.msg`.[1]

Edite o arquivo `RoverEvents.msg` e coloque exatamente:

```
std_msgs/String info           # Informação sobre descoberta, problema ou ação
geometry_msgs/Pose rover_location  # Posição do rover no momento do evento
```

- `std_msgs/String info`: define um campo chamado `info` do tipo `std_msgs/String`, ou seja, um texto com metadados sobre o que aconteceu.[1]
- `geometry_msgs/Pose rover_location`: define um campo chamado `rover_location` do tipo `geometry_msgs/Pose`, que contém posição (x, y, z) e orientação (quaternion).[1]

3. Ajustar o CMakeLists.txt

Abra `~/ros2_ws/src/custom_interfaces/CMakeLists.txt` e deixe no mínimo assim (adaptando o que já existe):

```
cmake_minimum_required(VERSION 3.8)
project(custom_interfaces)

if(CMAKE_COMPILER_IS_GNUCXX OR CMAKE_CXX_COMPILER_ID MATCHES "Clang")
  add_compile_options(-Wall -Wextra -Wpedantic)
endif()

# encontrar dependências
find_package(ament_cmake REQUIRED)
find_package(rclcpp REQUIRED)
find_package(std_msgs REQUIRED)
find_package(geometry_msgs REQUIRED)
find_package(rosidl_default_generators REQUIRED)

if(BUILD_TESTING)
  find_package(ament_lint_auto REQUIRED)
  set(ament_cmake_copyright_FOUND TRUE)
  set(ament_cmake_cpplint_FOUND TRUE)
  ament_lint_auto_find_test_dependencies()
endif()

# gerar as interfaces (mensagens) deste pacote
rosidl_generate_interfaces(${PROJECT_NAME}
  "msg/RoverEvents.msg"
  DEPENDENCIES std_msgs geometry_msgs
)

ament_package()
```

- `find_package(...)`: diz ao CMake que este pacote depende de `ament_cmake`, `rclcpp`, `std_msgs`, `geometry_msgs` e `rosidl_default_generators` para conseguir compilar a mensagem.[1]
- `rosidl_generate_interfaces(${PROJECT_NAME} "msg/RoverEvents.msg" DEPENDENCIES std_msgs geometry_msgs)`: manda o sistema de build gerar código para a mensagem `RoverEvents.msg`, informando que ela usa tipos dos pacotes `std_msgs` e `geometry_msgs`.[1]

4. Ajustar o package.xml

Abra `~/ros2_ws/src/custom_interfaces/package.xml` e deixe algo como:

```
<?xml version="1.0"?>
<?xml-model href="http://download.ros.org/schema/package_format3.xsd" schematypens="http:
<package format="3">
  <name>custom_interfaces</name>
  <version>0.0.0</version>
  <description>Custom interfaces for Mars rover events</description>
  <maintainer email="ubuntu@todo.todo">ubuntu</maintainer>
  <license>TODO: License declaration</license>

  <buildtool_depend>ament_cmake</buildtool_depend>

  <depend>rclcpp</depend>
  <depend>std_msgs</depend>
  <depend>geometry_msgs</depend>

  <build_depend>rosidl_default_generators</build_depend>

  <exec_depend>rosidl_default_runtime</exec_depend>
  <member_of_group>rosidl_interface_packages</member_of_group>

  <test_depend>ament_lint_auto</test_depend>
  <test_depend>ament_lint_common</test_depend>

  <export>
    <build_type>ament_cmake</build_type>
  </export>
</package>
```

- As tags `<depend>` indicam quais pacotes são usados em tempo de compilação e execução (`rclcpp`, `std_msgs`, `geometry_msgs`).[1]
- `<build_depend>rosidl_default_generators</build_depend>` e `<exec_depend>rosidl_default_runtime</exec_depend>` ativam o sistema de geração de mensagens e fazem o runtime reconhecer as novas mensagens.[1]

5. Compilar, dar source e testar

Na raiz do workspace:

```
cd ~/ros2_ws
colcon build --packages-select custom_interfaces
source install/setup.bash
```

- `colcon build --packages-select custom_interfaces`: compila só o seu pacote `custom_interfaces` e gera o código da nova mensagem.[1]
- `source install/setup.bash`: atualiza as variáveis de ambiente para o terminal enxergar a mensagem recém-gerada; se não fizer isso, ROS2 não encontra o tipo `custom_interfaces/msg/RoverEvents`.[1]

Para verificar se deu certo:

```
ros2 interface show custom_interfaces/msg/RoverEvents
```

- Este comando imprime a estrutura da mensagem, mostrando `std_msgs/String info` (com o campo `string data` interno) e `geometry_msgs/Pose rover_location` (com `Point position` e `Quaternion orientation`).[1]
- Se essa estrutura aparecer, a interface está criada e pronta para ser usada em nós ROS2 (publicadores/assinantes).[1]

Se você quiser, na próxima mensagem pode mandar um print de erro ou o conteúdo atual do `CMakeLists.txt`/`package.xml` que a explicação pode ir linha por linha em cima do que você já tem.

⁂

1. screencapture-app-theconstruct-ai-Desktop-2025-12-15-13_05_15.pdf