Ros2_4w (Action)

- **ACTION (Topic + Service 혼합 개념)**

  물리적 동작 실행 으로 시간 대기 필요 , 액션 Client 에서 이동 좌표 지시 하면 액션 서버는 이동 하며 실시간 좌표(Action Feedback)를 Client에게 전송하며 완료 결과의 **Complete/Failure**를 전송

- Action Massage Flow_ Action은 5개 형태로 분류 된다

1. Action Goal : Action Client 가 Action server에게 요청 (Request) , Server가 Client에게 Response
2. Action Feedback : Action Server에서 실시간 Data를 Client에게 Feedback (1번 동기)
3. Action GoalStatus : 실제 위치/좌표 까지 이동 하는 상태의 F/B (비동기)
4. Action CancelGoal : Action은 진행중 시간적 송가 발생 하는데 진행중 경로 폐쇄 & 명령에러 발생시 Action Goal에 대한 취소를 진행
5. Action Result : Action 취소 / 실행 완료에 대한 결과를 F/B

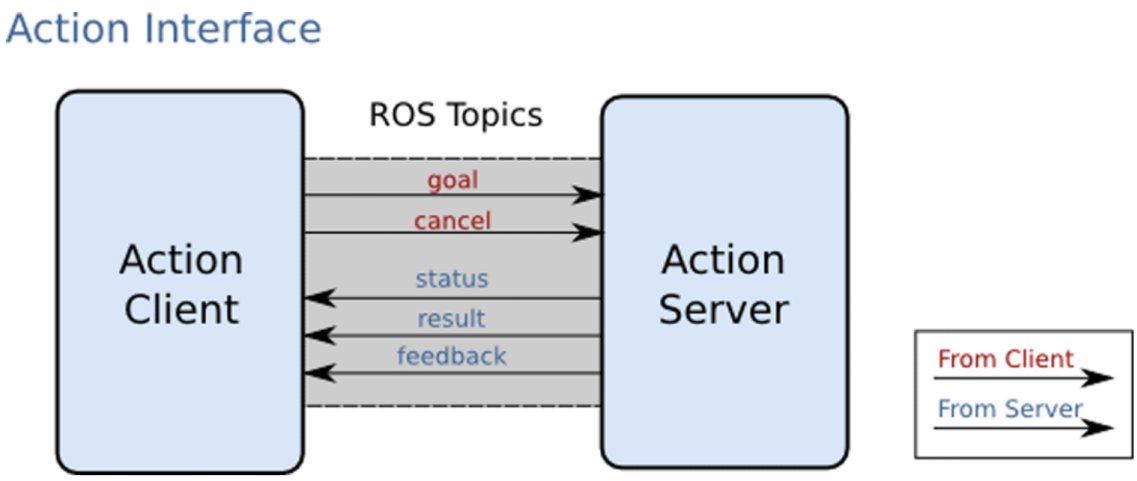


image from : The Construct

1. **Turtlesim(Node)를 작성하여 Action 목표 전달 확인 (Client , Server 정보)**

shin@ubuntu:~$ ros2 node info /turtlesim

Action Servers:
/turtle1/rotate_absolute: turtlesim/action/RotateAbsolute

Action Clients:

—> turtlesim은 Server(subscribe) 임.

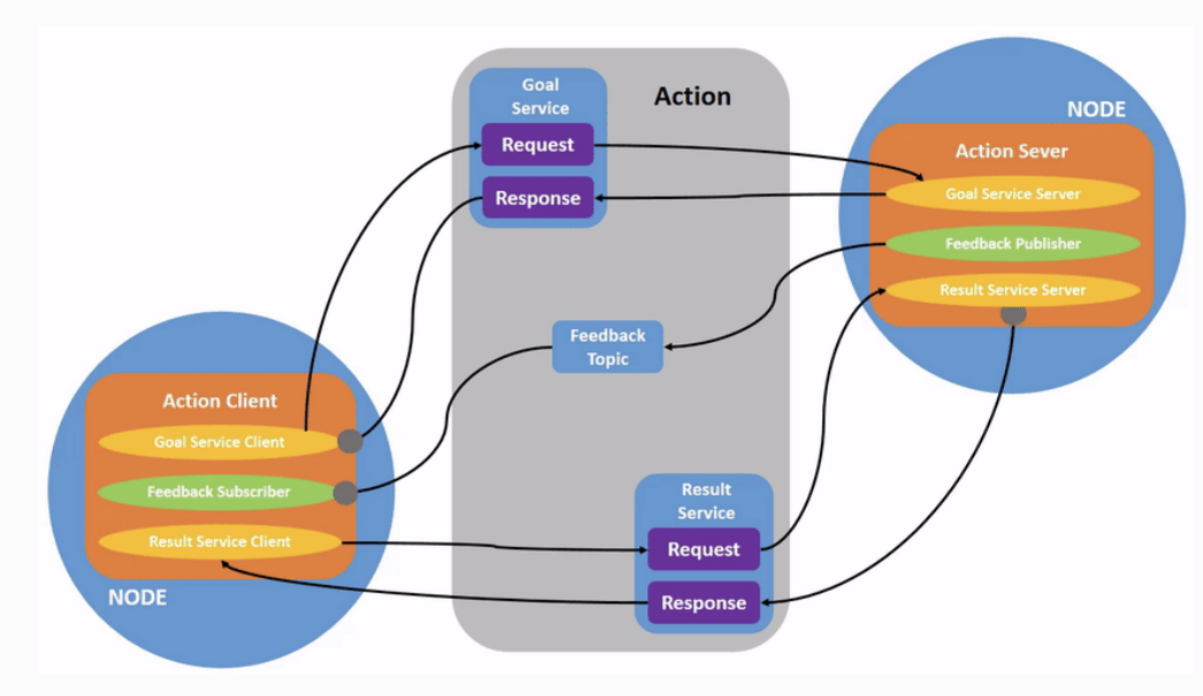
shin@ubuntu:~$ ros2 node info /teleop_turtle

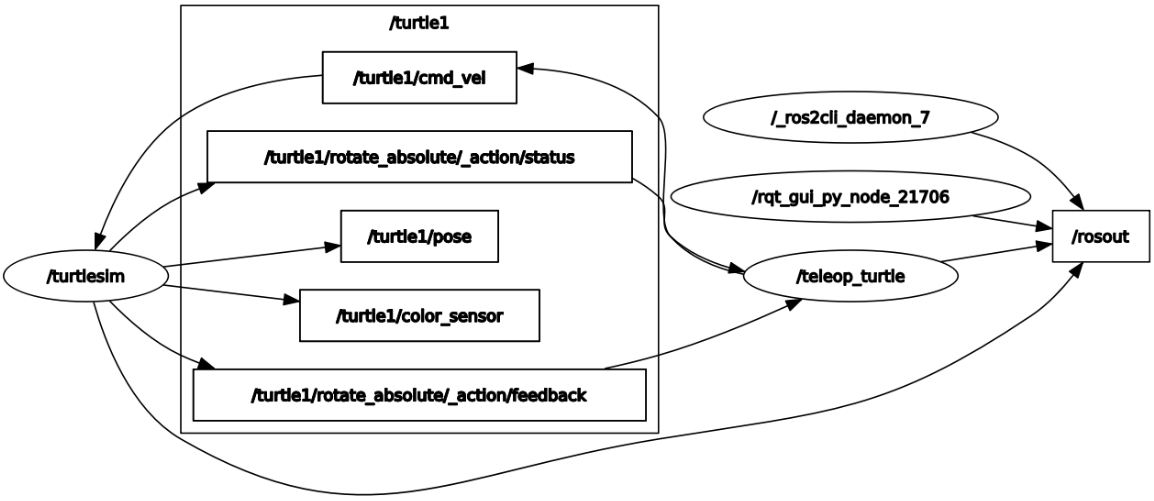
Action Servers:

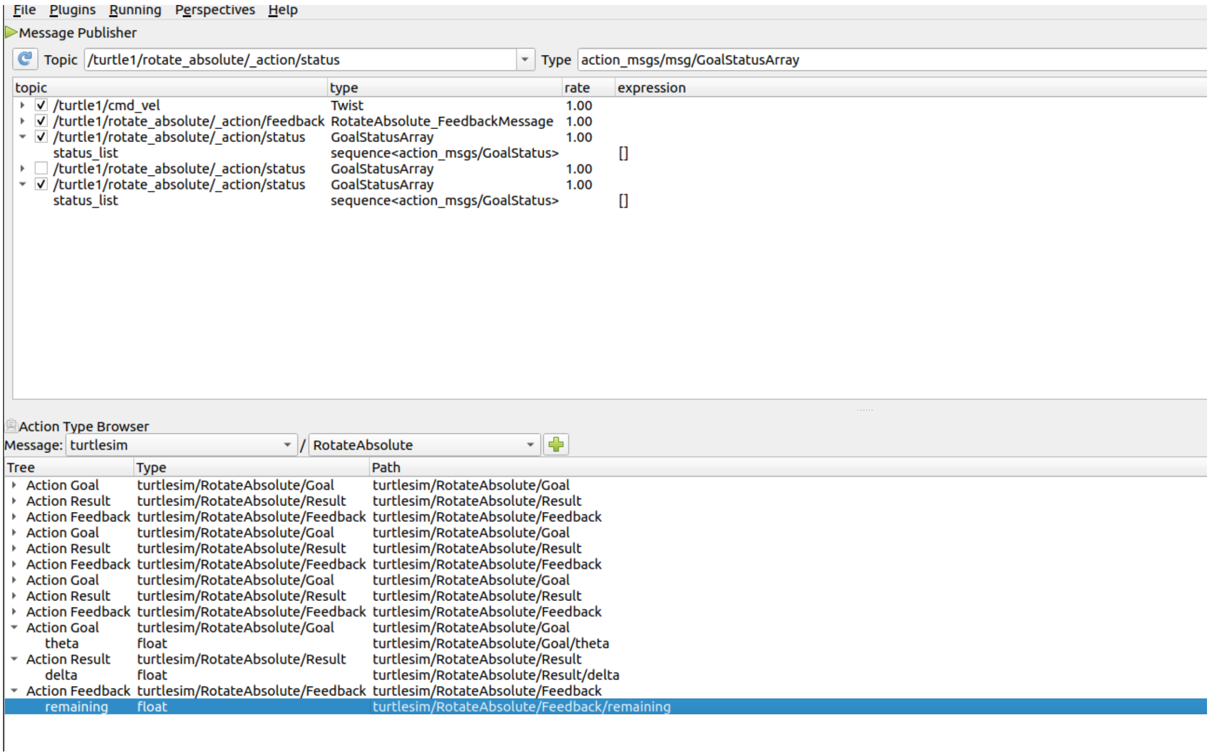
Action Clients:
/turtle1/rotate_absolute: turtlesim/action/RotateAbsolute

—> **teleop_turtle은 Client(publisher)임**

shin@ubuntu:~$ ros2 action info /turtle1/rotate_absolute
Action: /turtle1/rotate_absolute
Action clients: 1
/teleop_turtle
Action servers: 1
/turtlesim







shin@ubuntu:~$

2. **Ros2 명령어를 통한 Action goal 전달**

명령어 : Ros2 action send_goal <action_name> <action_type> “<values>”

1. ros2 action send_goal **/turtle1/rotate_absolute** ***turtlesim/action/RotateAbsolute*** **"{theta: 1.5708}”**

Result:
delta: -0.544001042842865

Goal finished with status: ABORTED (미달된 delta로 ABORTED) —> 재실행 (동일 명령어)

Result:
delta: -0.016000032424926758

Goal finished with status: **SUCCEEDED**

명령어 : Ros2 action send_goal <action_name> <action_type> “<values>” —feedback

남은 Data를 Feedback 표시하는 명령

Feedback:
remaining: 0.5356224775314331

Result:
delta: -0.6559992432594299

Goal finished with status: ABORTED —> 재실행 (동일 명령어)

Feedback:
remaining: 0.007621407508850098

Result:
delta: -0.5120010375976562

Goal finished with status: SUCCEEDED

ROS 2 interface

Interface

- ROS 노드간에 데이터를 주고 받기 위해 토픽, 서비스, 액션 3가지 유형중 하나로 인터페이스 통신을 하고 이러한 인터페이스를 설명하기 위해 단순화된 설명 언어인 , IDL(Interface Definition Language)을 사용 합니다
- 토픽 , 서비스, 액션은 msg , srv , action interface를 사용하고 정수, 부동 소수점, 불리언 기본으로 한다

Interface 종류

메세지 interface

1. 단순 자료형

https://github.com/ros2/common_interfaces/tree/foxy/**std_msgs**

정수(integer) , 부동 소숫점 (floating point) , 불린(boolean)

2. 메세지안에 메세지를 품고 있는 데이터 구조

https://github.com/ros2/common_interfaces/blob/foxy/geometry_msgs/msg/Twist.msg

ex) geometry_msgs/msgs/Twist

- **Field Type**

fieldtype1 fieldname1
fieldtype2 fieldname2
fieldtype3 fieldname3

ex) **int32** my_int
string my_string

Type name	C++	Python	DDS Type
bool	bool	builtins.bool	boolean
byte	unit8_t	builtins.bytes	octet
char	char	builtins.str	char
float32	float	builtins.float	float
float64	double	builtins.float	double
int8	int8_t	builtins.int	octet
unit8	unit8_t	builtins.int	octet
int16	int16_t	builtins.int	short
unit16	unit16_t	builtins.int	unsigned short
int32	int32_t	builtins.int	long
unit32	unit32_t	builtins.int	unsigned long
int64	int64_t	builtins.int	long long
unit64	unit64_t	builtins.int	unsigned long long
string	std::string	builtins.str	string
wstring	std::u16string	builtins.str	wstring
static array	std::array<T,N>	builtins.list	T[N]
unbounded dynamic array	std::vector	builtins.list	sequence
bounded dynamic array	custom_class<T,N>	builtins.list	builtins.list<T,N>
bounded string	std::string	builtins.str	string

- **Field name**

필드 이름은 단어를 구분하기 위해 밑줄이 있는 소문자 영숫자여야 하고 알파벳 문자로 시작해야 하며 밑줄로 끝나거나 두 개의 연속 밑줄이 없어야 합니다.

ex)

int32[] unbounded_integer_array
int32[5] five_integers_array
int32[<=5] up_to_five_integers_array

string string_of_unbounded_size
string<=10 up_to_ten_characters_string

string[<=5] up_to_five_unbounded_strings
string<=10[] unbounded_array_of_strings_up_to_ten_characters_each
string<=10[<=5] up_to_five_strings_up_to_ten_characters_each

- **Field default**

기본값은 메시지 유형의 모든 필드로 설정할 수 있고. 현재 기본값은 **string형 배열 및 복합 형식**

사용 불가

기본값을 정의하는 것은 필드 정의 줄에 세 번째 요소를 추가하여 수행됩니다.

ex) fieldtype1 fieldname1 fielddefaultvalue

```
uint8 x 42
int16 y -2000
string full_name "John Doe"
int32[] samples [-200, -100, 0, 100, 200]
```

- Constant

  각 상수 정의는 기본값이 있는 필드 설명과 같지만 이 값은 프로그래밍 방식으로 변경할 수 없고

  이 값 할당은 등호 '=' 기호를 사용하여 표시한다

  constanttype CONSTANTNAME = constantvalue

ex)

```
int32 X=123
int32 Y=-123
string FOO="foo"
string EXAMPLE='bar'
```

- Service interface

서비스는 클라이언트(요청자)가 서버(응답자)가 짧은 계산을 수행하고 결과를 반환하기를 기다리는 요청/응답 통신이고 `.srv srv/` 파일이다

```
string str
---
string str
```

```
# request constants
int8 FOO=1
int8 BAR=2
# request fields
int8 foobar
another_pkg/AnotherMessage msg
---
# response constants
uint32 SECRET=123456
# response fields
another_pkg/YetAnotherMessage val
CustomMessageDefinedInThisPackage value
uint32 an_integer
```

- **Action Interface**

Action은 장기 실행 요청/응답 통신으로, 작업 클라이언트(요청자)는 작업 서버(응답자)가 작업을 수행하고 결과를 반환할 때까지 대기합니다. 서비스와 달리 작업은 오래 실행되고(몇 초 또는 몇 분) 발생하는 동안 피드백을 제공할 수 있으며 중단될 수 있습니다.

```
<request_type> <request_fieldname>
---
<response_type> <response_fieldname>
---
<feedback_type> <feedback_fieldname>
```

```
int32 order
---
int32[] sequence
---
int32[] sequence
```

토픽 , 서비스 , 액션 비교

	토픽	서비스	액션
연속성	연속성	일회성	복합(토픽+서비스)
방향성	단방향	양방향	양방향
동기성	비동기	동기	동기+비동기
다자간 연결	1:N , 1:1 , N:1 , N:N (Publisher:Subscribe)	1:1 (Server : Client)	1:1 (Server : Client)
노드 역할	Publisher & Subscriber	Client & Server	Client & Server
동작 트리거	Publisher	Client	Client

인터페이스	msg 인터페이스	srv 인터페이스	action 인터페이스
CLI 명령어	ros2 topic , ros2 interface	ros2 service ros2 interface	ros2 action ros2 interface
사용 예	센서 데이타, 로봇 상태 로봇좌표, 로봇속도 명령등	LED 제어, 모터토크on/off IK/FK계산, 이동경로 계산	목적지로 이동, 물건파지 복합테스트 등

	msg 인터페이스	srv 인터페이스	action 인터페이스
확장자	*.msg	*.srv	*.action
테이터	토픽 데이타(data)	서비스 요청(request) - - - 서비스 응답 (response)	액션 목표(goal) - - - 액션 결과(result) - - - 액션 피드백(feedback)
형식	fieldtype1 fieldname1 fieldtype2 fieldname2 fieldtype3 fieldname3	fieldtype1 fieldname1 fieldtype2 fieldname2 - - - fieldtype3 fieldname3 fieldtype4 fieldname4	fieldtype1 fieldname1 fieldtype2 fieldname2 - - - fieldtype3 fieldname3 fieldtype4 fieldname4 - - - fieldtype5 fieldname5 fieldtype6 fieldname6
사용 예	[geometry_msgs/msgs/Twist]	[turtlesim/srv/Spawn.srv]	[turtlesim/action/RotateAbsolute.action]
	Vector3 linear Vector3 angular	float32 x float32 y float32 theta string name - - - string name	float32 theta - - - float32 delta - - - float32 remaining

ROS2 파라미터

- 파라미터는 서비스 통신 방법을 통하여 NODE 내부 또는 외부에서 NODE내 파라미터를 SET(지정), GET(가져오는것)해서 사용하는 목적이다
- 파라미터는 RCL(ROS Client Libraries)의 기본기능으로 모든 NODE가 Parameter server를 가지고 있고 Parameter Client도 가질수 있어 NODE 내에 파라미터를 읽고 쓸수 있다

- **turtlesim package의 Parameter list 명령어**

~$ ros2 param list

/teleop_turtle:
scale_angular
scale_linear
use_sim_time
/turtlesim:
background_b
background_g
background_r
use_sim_time

- **ros2 param get <node_name> <parameter_name> : 현재Node에 설정된 Parameter 가져오기**

~$ ros2 param get /turtlesim background_b
설정 Value 확
Integer value is: 255

Integer value is: 86

Integer value is: 69

- **ros2 param set <node_name> <parameter_name> <value> Parameter 변경 설정**
- NODE Clear시 초기 data 원복

~$ ros2 param set /turtlesim background_r 150 : 배경색 변경

- **ros2 param dump <node_name> 설정(set)된 Parameter 저장 명령어**

~$ ros2 param dump /turtlesim

/turtlesim:
ros__parameters:
background_b: 255
background_g: 86
background_r: 150 (변경 Parameter)
use_sim_time: false

- **ros2 param load <node_name> <parameter_file> 기존에서 load하는 명령**

~$ ros2 param load /turtlesim turtlesim.yaml

Set parameter background_b successful
Set parameter background_g successful

Set parameter background_r successful
Set parameter use_sim_time successful

- **ros2 run <package_name> <executable_name> - -ros - args - -params-file <file_name>**

~$ ros2 run turtlesim turtlesim_node --ros-args --params-file turtlesim.yaml

Node 생성시 부터 변경된 Parameter 가 적용된 file 생성