Parameter Server

⇒ A parameter server is a shared, multi-variate dictionary that is accessible via network APIs.

- Nodes uses this server to store and retrieve parameters at runtime.
- As it is not designed for high-performance, it is best used for static, non-binary data such as configuration parameters.
- Parameter Server runs inside the ROS Master.

Example:

/cam/left/name :	leftcam
/cam/left/exp :	1
/cam/right/name :	rightcam
/cam/right/exp :	1.1

{this is a dictionary (dic of dic)}

/cam → left: {name : leftcam, exp: 1}
right: {name: rightcam, exp: 1.1}

{this is dictionary of dictionary}

/cam/left → name: leftcam
exp: 1 → {this is dictionary}

Scanned by CamScanner

* Parameter type

⇒ The parameter Server uses XMLRPC data types for parameter values, which include:

① 32-bit integer
② booleans
③ strings
④ doubles
⑤ iso8601 data
⑥ lists
⑦ base64-encoded binary data.

→ The parameter Server represents ROS namespaces as dictionaries.

* Parameter Tools (rosparam)

⇒ There are also special converters for angle radian/degree representation.

angle1: rad(2*Pi)	angle1: !degree 180.0
angle2: deg(180)	angle2: !radians 3.14165

⇒ In either case the angle value is converted to radians (float)

⇒ The <rosparam> tag can be put inside of a <node> tag, in which case the parameter is treated like a private name.

① rosparam list

rosparam list
⟶ lists all the parameter names.

rosparam list /namespace

→ {Lists all parameters in a Particular namespace}

② rosparam get

rosparam get parameter-name

→ returns parameter value

③ rosparam set

rosparam set parameter_name value

Example: /gain "p: 1.0 i: 1.0 d: 1.0"

↓

using YAML dictionary.

④ rosparam delete

rosparam delete parameter_name

⑤ rosparam dump

rosparam dump dump.yaml

→ Dump the YAML-formatted contents of the Parameter Server to a file.

rosparam dump dump.yaml /namespace

→ Dump only the parameter in the specified namespace.

⑥ rosparam load

```
rosparam load dump.yaml
```

→

Load parameters from a YAML file into the specified [namespace] (default /)

ROSCPP (Parameter Server)

⇒ roscpp has two different parameter APIs: the "bare" versions which live in the ros::param name space and the handle versions which are called through the ros::NodeHandle interface.

① Getting parameter

ⓐ ros::NodeHandle::getParam()

```
nh.getParam("/global_name", global_name);
```

returns true if it exist.

```
nh.getParam("relative_name", relative_name);
```

```
nh.param<std::string>("default_param"
        , default_param, "default_value");
```

If the parameter does not exist default value will be chosen.

(b) ros:: Param:: get ()

ros :: Param:: get ("/global_name", global_name);

ros:: Param :: get ("relative_Name", relative_name);

ros:: Param:: param<std::string>("default_param",
default_param,
"default_Value");

(2) Setting Parameters

(a) ros:: NodeHandle :: setParam ();

(b) ros :: param : set ();

(3) Checking parameter Existence

(a) ros :: NodeHandle :: hasParam ();

nh. hasParam ("my-Param")

(b) ros :: param :: has ();

ros :: param:: has ("my-Param")

(4) Deleting Parameters

(a) ros:: NodeHandle :: deleteParam ()

nh. deleteParam ("my-Param")

(b) ros :: param :: del ("my-Param")

⑤ Accessing Private Parameters

ⓐ
```
ros::NodeHandle nh("~");
std:: string param.
nh.getParam("private_name", param);
```

ⓑ
```
std:: string param;
ros:: param :: get ("~private_name", param);
```

⑥ Searching for Parameter Keys:

ⓐ `ros::NodeHandle::searchParam()`

ⓑ `ros :: param :: search ()`

⇒ Search parameter from the closest name space.

lists ⟶ std:: vector

dictationario ⟶ std:: map