Note: This tutorial assumes that you have completed the previous tutorials: Comprendre les Topics ROS (/fr/ROS/Tutorials/UnderstandingTopics).

Please ask about problems and questions regarding this tutorial on answers.ros.org (http://answers.ros.org). Don't forget to include in your question the link to this page, the versions of your OS & ROS, and also add appropriate tags.

Comprendre les services et paramètres ROS

Description: Ce tutoriel introduit les concepts de services et de paramètres sous ROS, ainsi que l'utilisation des outils en ligne de commande rosservice (/rosservice) et rosparam (/rosparam).

Tutorial Level: BEGINNER

Next Tutorial: Using rqt_console and roslaunch (/ROS/Tutorials/UsingRqtconsoleRoslaunch)

Sommaire

En considérant que votre noeud `turtlesim_node` du tutoriel précédent est toujours en cours d'exécution, regardons quels services turtlesim fournit :

1. Services ROS

Les services sont un autre moyen de communication entre noeuds. Les services permettent aux noeuds d'envoyer une **requête** et de recevoir une **réponse**.

2. Utilisation de rosservice

`rosservice` permet de facilement mettre en oeuvre des services connectés aux clients ou services du framework ROS. `rosservice` offre de nombreuses commandes qui peuvent être utilisées sur les services, comme indiqué ci-dessous :

Usage :

```
rosservice list         Affiche les informations sur les services actifs
rosservice call         Appelle le service avec les arguments fournis
rosservice type         Affiche le type du service
rosservice find         Cherche les services à partir de leur type
rosservice uri          Affiche l'uri du service ROSRPC
```

2.1 rosservice list

```
$ rosservice list
```

La commande `list` montre que le noeud turtlesim fournit 9 services : `reset`, `clear`, `spawn`, `kill`, `turtle1/set_pen`, `/turtle1/teleport_absolute`, `/turtle1/teleport_relative`, `turtlesim/get_loggers`, et `turtlesim/set_logger_level`. Par ailleurs, il y a 2 services relatifs au noeud `rosout` : `/rosout/get_loggers` et `/rosout/set_logger_level`.

```
/clear
/kill
/reset
/rosout/get_loggers
/rosout/set_logger_level
/spawn
/teleop_turtle/get_loggers
/teleop_turtle/set_logger_level
/turtle1/set_pen
/turtle1/teleport_absolute
/turtle1/teleport_relative
/turtlesim/get_loggers
/turtlesim/set_logger_level
```

Examinons plus en détails le service `clear` au moyen de `rosservice type`:

2.2 rosservice type

Usage:

```
rosservice type [service]
```

Trouvons quel est le type du service `clear` :

```
$ rosservice type /clear
```

```
std_srvs/Empty
```

Ce service est vide, ce qui signifie que l'appel du service ne prend aucun argument (c'est à dire qu'aucune donnée n'est envoyée par la **requête** et qu'aucune donnée n'est reçue lors de la réception de la **réponse**). Appelons ce service au moyen de `rosservice call`:

2.3 rosservice call

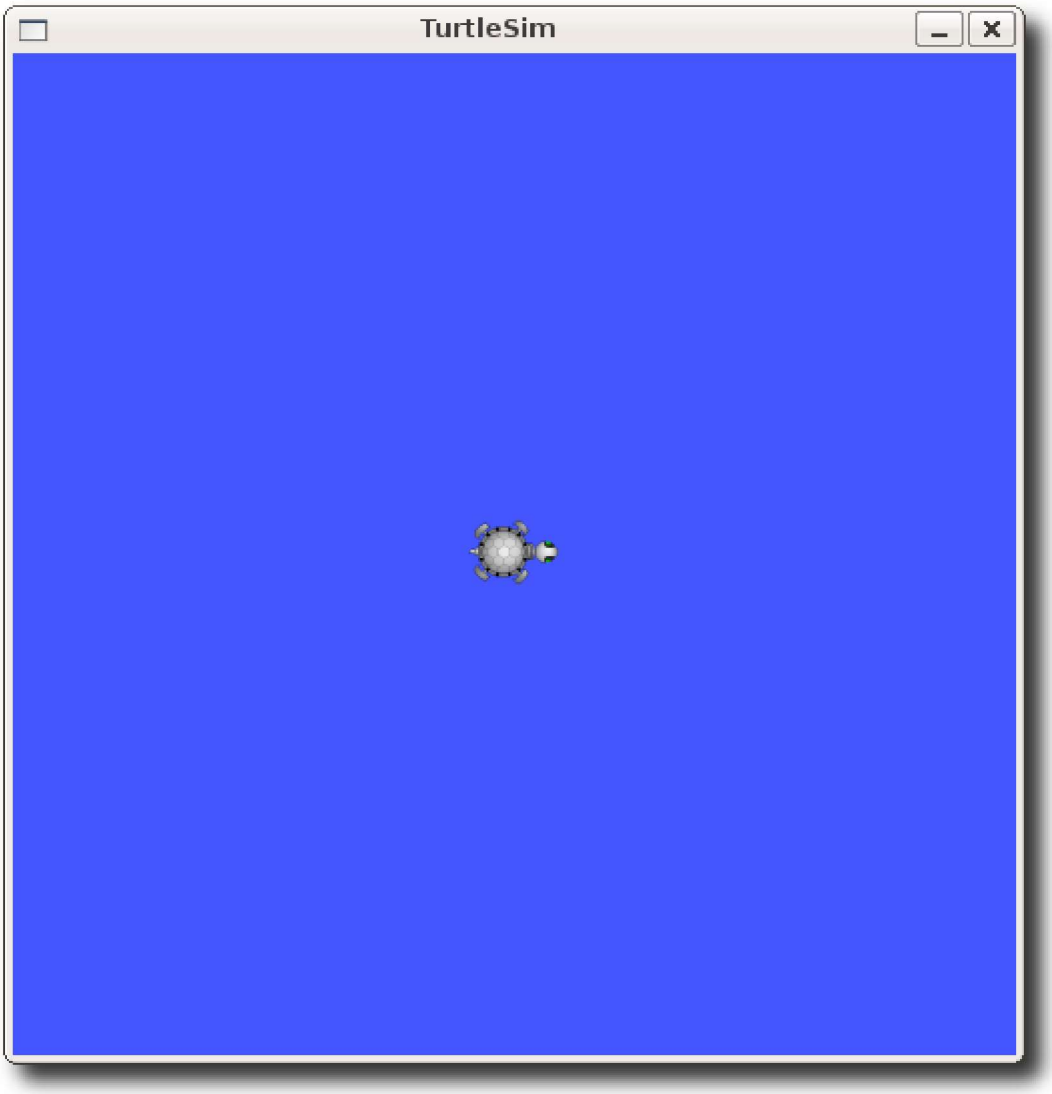
Usage:

```
rosservice call [service] [args]
```

De ce cas, le service est appelé sans arguments, puisque le type du service est vide :

```
$ rosservice call /clear
```

Comme attendu, l'arrière-plan du noeud `turtlesim_node` est effacé.



Voyons le cas où un service possède des arguments, en examinant les informations du service spawn :

```
$ rosservice type /spawn | rossrv show
```

```
float32 x
float32 y
float32 theta
string name
---
string name
```

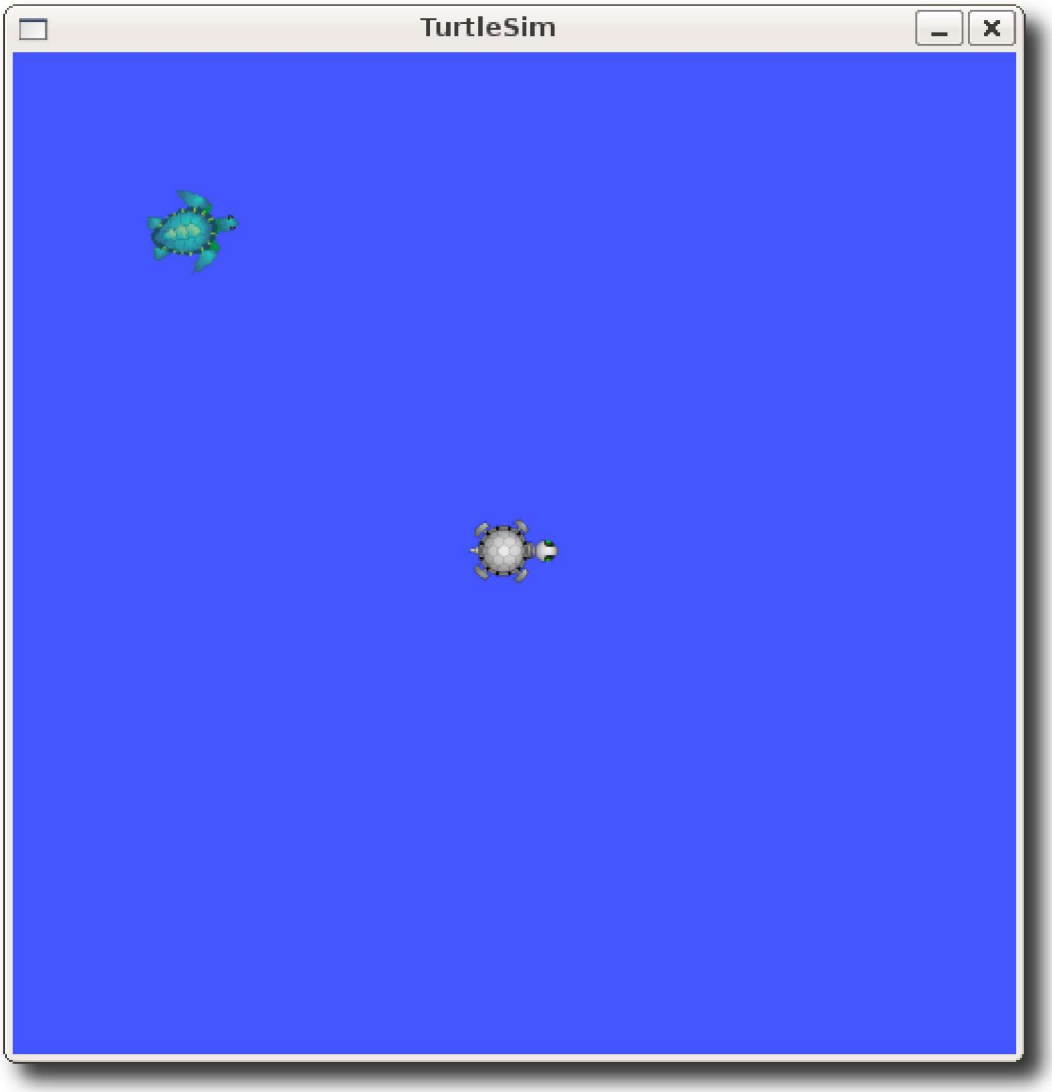
Ce service permet de pondre une nouvelle tortue à l'emplacement et suivant l'orientation donnée. Le champ nom est optionnel, aussi ne donnons pas de nom à notre tortue et laissons turtlesim le mettre pour nous.

```
$ rosservice call /spawn 2 2 0.2 ""
```

L'appel du service renvoie le nom de la nouvelle tortue.

```
name: turtle2
```

A présent, turtlesim doit resembler à cela :



3. Utilisation de rosparam

`rosparam` permet de stocker et de maniipuler des données dans le Serveur de Paramètres (/Parameter%20Server) de ROS. Le Serveur de Paramètres peut stocker des types entiers, flottants, booléens, dictionnaires et listes. rosparam utilise la syntaxe YAML. Dans les cas les plus

simples, YAML est très lisible : `1` est un entier, `1.0` est un flottant, `one` est une chaîne, `true` est un booléen, `[1, 2, 3]` est une liste d'entiers, et `{a: b, c: d}` est un dictionnaire. `rosparam` possède plusieurs commandes qui peuvent être utilisées sur les paramètres, comme montré ci-dessous :

Usage :

```
rosparam set            définit un paramètre
rosparam get            lit un paramètre
rosparam load           charge des paramètres depuis un fichier
rosparam dump           écrit des paramètres dans un fichier
rosparam delete         supprime un paramètre
rosparam list           liste les noms des paramètres
```

Voyons quels paramètres sont actuellement sur le serveur de paramètres :

3.1 rosparam list

```
$ rosparam list
```

On peut voir ici que, sur le serveur de paramètres, le noeud turtlesim a 3 paramètres concernant la couleur de fond :

```
/rosdistro
/roslaunch/uris/host_nxt__43407
/rosversion
/run_id
/turtlesim/background_b
/turtlesim/background_g
/turtlesim/background_r
```

Changeons l'un des paramètres avec `rosparam set`:

3.2 rosparam set et rosparam get

Usage:

```
rosparam set [param_name]
rosparam get [param_name]
```

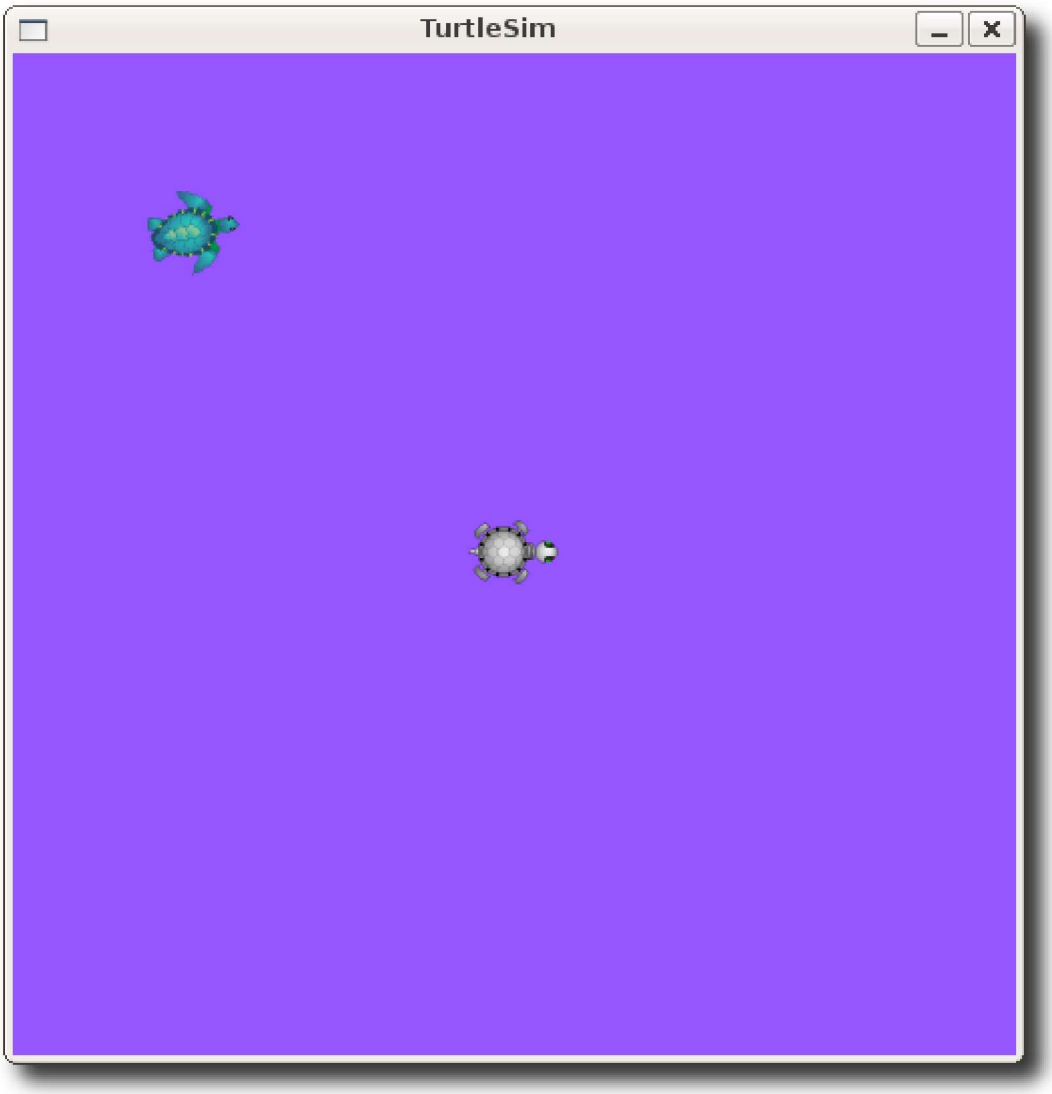
Ceci va changer la composante rouge de la couleur de fond :

```
$ rosparam set /turtlesim/background_r 150
```

La valeur du paramètre a changé, à présent il faut appeler le service clear pour que le changement du paramètre prenne effet :

```
$ rosservice call /clear
```

A présent, turtlesim ressemble à cela :



Regardons maintenant les valeurs des autres paramètres sur le serveur de paramètres. Obtenons la valeur de la composante verte du fond :

```
$ rosparam get /turtlesim/background_g
```

```
86
```

On peut aussi utiliser `rosparam get /` pour nous montrer tout le contenu du serveur de paramètres.

```
$ rosparam get /
```

```
rosdistro: 'noetic

  '
roslaunch:
  uris:
    host_nxt__43407: http://nxt:43407/
rosversion: '1.15.5

  '
run_id: 7ef687d8-9ab7-11ea-b692-fcaa1494dbf9
turtlesim:
  background_b: 255
  background_g: 86
  background_r: 69
```

Vous pouvez souhaiter enregistrer ce résultat dans un fichier, afin de le recharger plus tard. Cela se fait facilement avec `rosparam`:

3.3 rosparam dump and rosparam load

Usage:

```
rosparam dump [file_name] [namespace]
rosparam load [file_name] [namespace]
```

Ici, nous allons enregistrer tous les paramètres dans le fichier params.yaml

```
$ rosparam dump params.yaml
```

Vous pouvez même charger le fichier yaml dans un nouvel espace de nom, par exemple `copy_turtle`:

```
$ rosparam load params.yaml copy_turtle
$ rosparam get /copy_turtle/turtlesim/background_b
```

```
255
```

Maintenant que vous avez compris le fonctionnement des services et paramètres, expérimentez l'utilisation de rqt_console et roslaunch (/ROS/Tutorials/UsingRqtconsoleRoslaunch)

Wiki: fr/ROS/Tutorials/UnderstandingServicesParams (dernière édition le 2020-12-15 16:33:27 par SMougel (/SMougel))

(https://www.openrobotics.org/)