ورقة بحثية مجلة مفتوحة

# سلسلة تعلم استخدام نظام ROS2 في نظام التشغيل Windows10 مباشرة (الجزء الثالث)

د. عيسى الغنام* ، د. إياد حاتم**

*(كلية الهندسة، جامعة المنارة ، البريد الإلكتروني: Essa.Alghannam@manara.edu.sy)

**(كلية الهندسة، جامعة المنارة، البريد الإلكتروني: iyadhatem@manara.edu.sy)

**الملخص**

تعد الموضوعات Topics والخدمات services عنصرًا حيويًا في ROS حيث تستخدم لتبادل المعلومات والاتصال بين العقد nodes. تعتمد الخدمات Services على نموذج الطلب والاستجابة call-and-response model، مقابل ذلك تعتمد الموضوعات على نموذج الناشر والمشترك publisher-subscriber model. تسمح الموضوعات للعقد بالنشر والاشتراك في تدفقات البيانات والحصول على تحديثات مستمرة، بينما تقدم "عقدة مزودة" بيانات عبر الخدمات فقط عندما يتم استدعاء الخدمة من قبل "عقدة عميل" معينة تحديدا. في هذا الجزء، سنتكلم عن مفهوم الموضوعات topics والخدمات services وآلية التعامل معها من خلال المحاكي Turtlesim في ROS2 وكذلك باستخدام الأداة rqt_graph وأيضا من خلال استخدم command line tools.

**كلمات مفتاحية** – ROS2, SERVICES, TURTLESIM, RQT, TOPICS, NODES.

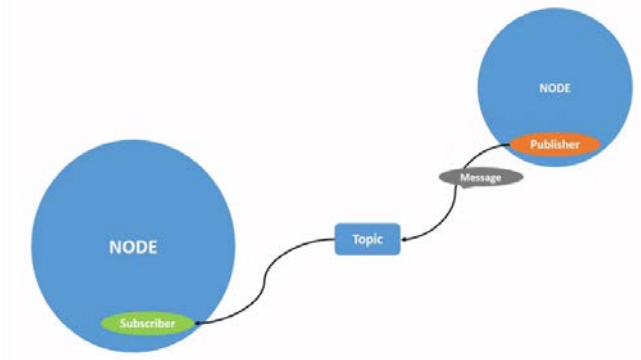
**Abstract:**

Topics and services are a vital component of ROS as they are used to exchange information and communicate between nodes. Services are based on the call-and-response model, while Topics are based on the publisher-subscriber model. Topics allow nodes to publish and subscribe to data streams and obtain continuous updates, while a “provider node” delivers data via services only when the service is called by a specifically designated “client node.” In this part, we will talk about the concept of topics and services and the mechanism for dealing with them through the Turtlesim emulator in ROS2, as well as using the rqt_graph tool, and also through using the command line tools.

**Keywords**: ROS2, SERVICES, TURTLESIM, RQT, TOPICS, NODES.

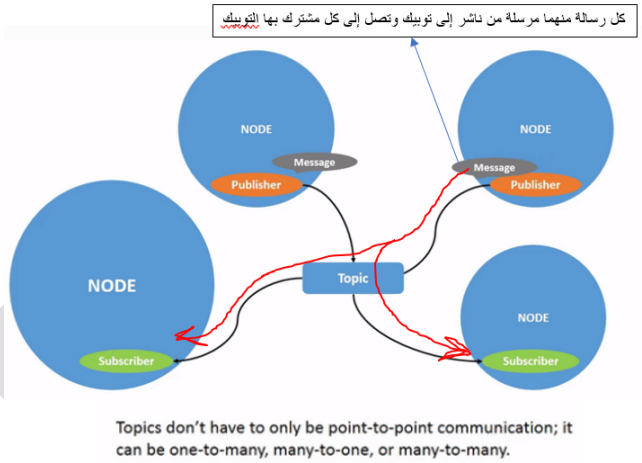
## I. الموضوعات TOPICS في ROS2

يقسم ROS2 الأنظمة المعقدة complex systems إلى العديد من العقد المعيارية modular nodes. تعد الموضوعات Topics عنصرًا حيويًا في ROS حيث تعمل بمثابة (ناقل bus) بين العقد لتبادل الرسائل messages.




**الشكل 1 . تواصل عقدتين عبر موضوع**

قد تنشر publish العقدة البيانات إلى أي عدد من الموضوعات number of topics ولها اشتراكات subscriptions في نفس الوقت simultaneously في أي عدد من الموضوعات to any number of topics.




**الشكل 2 . أنماط تواصل لعدة عقد عبر موضوع**

تعد الموضوعات إحدى الطرق الرئيسية التي يتم بها نقل البيانات بين العقد وبالتالي بين أجزاء مختلفة من النظام.

افتح طرفية جديدة وقم بتشغيل:

```
ros2 run turtlesim turtlesim_node
```

افتح طرفية أخرى أيضا وقم بتشغيل:

```
ros2 run turtlesim turtle_teleop_key
```

تذكر من الجزء السابق أن أسماء هذه العقد هي: /turtlesim و /teleop_turtle افتراضيًا.

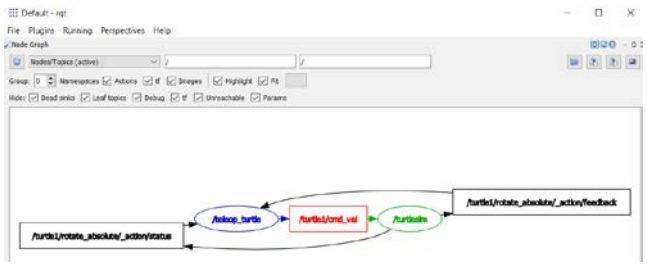
سنستخدم rqt_graph لإظهار العقد والموضوعات، فضلاً عن الروابط بينها. لتشغيل rqt_graph، افتح محطة طرفية جديدة وأدخل الأمر:

```
rqt_graph
```

أو:

```
rqt
```

والتي تعطي:



**الشكل 3 . استخدام rqt لإظهار العقد والموضوعات في بيئة العمل**

يجب أن ترى عقدتين بالأزرق والأخضر وموضوع topic بالأحمر أعلاه، بالإضافة إلى إجراءين actions في الرسم البياني باللون الأسود (دعنا نتجاهلهما الآن). إذا قمت بتمرير الماوس فوق الموضوع في المنتصف، فسترى اللون الأحمر عليه حيث يتم تمييزه كما في الصورة أعلاه.

يوضح الرسم البياني كيفية تواصل العقدة /turtlesim والعقدة /teleop_turtle مع بعضهما البعض حول موضوع ما. تقوم العقدة /teleop_turtle بنشر البيانات (ضغطات المفاتيح التي تدخلها لتحريك السلحفاة حولها) إلى الموضوع /turtle1/cmd_vel، ويتم اشتراك العقدة /turtlesim في هذا الموضوع لتلقي البيانات.

كما تعرفنا سابقا يعد استخدام rqt_graph وهو أداة رسومية، مفيداً للغاية عند فحص أنظمة أكثر تعقيدًا مع العديد من العقد والموضوعات المتصلة بعدة طرق مختلفة.

الآن سنلقي نظرة على بعض أدوات سطر الأوامر some command line tools لفهم الموضوعات topics.

سيؤدي تشغيل الأمر ros2 topic list في محطة طرفية جديدة إلى إرجاع قائمة بجميع الموضوعات النشطة حاليًا في النظام:

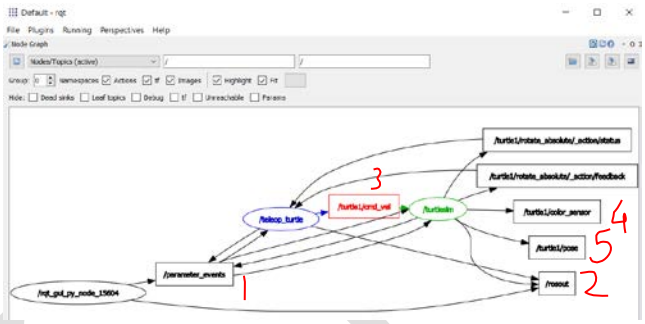
```
/parameter_events
/rosout
/turtle1/cmd_vel
/turtle1/color_sensor
/turtle1/pose
```

ستعيد ros2 topic list -t نفس قائمة الموضوعات، هذه المرة مع نوع الموضوع topic type مرفق بين قوسين:

```
/parameter_events
[rcl_interfaces/msg/ParameterEvent]
/rosout [rcl_interfaces/msg/Log]
/turtle1/cmd_vel [geometry_msgs/msg/Twist]
/turtle1/color_sensor [turtlesim/msg/Color]
/turtle1/pose [turtlesim/msg/Pose]
```

هذه attributes، تبين كيفية معرفة العقد بأنها تتحدث عن نفس المعلومات أثناء انتقال هذه المعلومات عبر الموضوعات.

إذا كنت تتساءل عن مكان كل هذه الموضوعات في rqt_graph، فيمكنك إلغاء تحديد جميع المربعات الموجودة ضمن Hide:



**الشكل 4 . إظهار كل الموضوعات والعقد المخفية افتراضيا في بيئة العمل في rqt**

في الوقت الحالي، اترك هذه الخيارات checked لتجنب الالتباس. للاطلاع على البيانات التي يتم نشرها حول موضوع ما، استخدم:

```
ros2 topic echo <topic_name>
```

نظرًا لأننا نعلم أن /teleop_turtle تنشر البيانات إلى /turtlesim عبر موضوع /turtle1/cmd_vel، فلنستخدم echo لفهم هذا الموضوع:

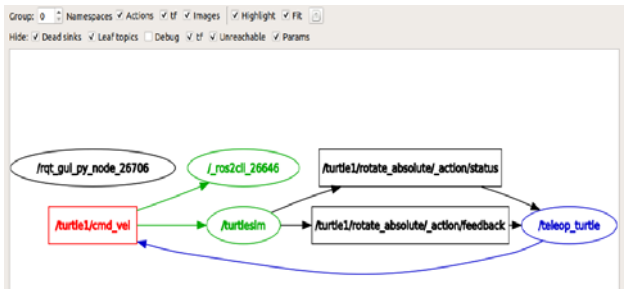
```
ros2 topic echo /turtle1/cmd_vel
```

في البداية ، لن يُرجع هذا الأمر أي بيانات. هذا لأنه ينتظر /teleop_turtle لنشر شيء ما.

عد إلى المحطة حيث يعمل turtle_teleop_key واستخدم الأسهم لتحريك السلحفاة. شاهد المحطة حيث يعمل echo في نفس الوقت، وسترى بيانات الموقع يتم نشرها لكل حركة تقوم بها:

```
linear:
  x: 2.0
  y: 0.0
  z: 0.0
angular:
  x: 0.0
  y: 0.0
  z: 0.0
  ---
```

عد الآن إلى rqt_graph وقم بإلغاء تحديد uncheck the Debug box.




**الشكل 5 . مخطط التواصل بين العقد بعد استخدام الأمر echo**

/_ros2cli_26646 هي العقدة التي تم إنشاؤها بواسطة أمر echo الذي قمنا بتشغيله للتو (قد يكون الرقم مختلفًا). يمكنك الآن أن ترى أن الناشر ينشر البيانات عبر موضوع cmd_vel، وأن هناك مشتركين فيه.

لا يجب أن تكون الموضوعات مجرد تواصل فردي؛ يمكن أن تكون رأس بأطراف one-to-many أو أطراف برأس many-to-one أو أطراف بأطراف many-to-many.

هناك طريقة أخرى للنظر إلى هذا وهي:

```
ros2 topic info /turtle1/cmd_vel
```

والتي تعيد:

```
Type: geometry_msgs/msg/Twist
Publisher count: 1
Subscription count: 2
```

العقد ترسل البيانات عبر الموضوعات باستخدام الرسائل. يجب على الناشرين والمشتركين إرسال واستقبال نفس النوع من الرسائل للتواصل.

أنواع الموضوعات التي رأيناها سابقًا بعد تشغيل ros2 topic list -t تتيح لنا معرفة نوع الرسالة المستخدمة في كل موضوع. تذكر أن موضوع cmd_vel له النوع:

```
geometry_msgs/msg/Twist
```

هذا يعني أنه يوجد في الحزمة geometry_msgs رسالة تسمى Twist.

الآن يمكننا تشغيل الأمر ros2 interface show <msg type> على هذا النوع لمعرفة تفاصيل أكثر عن نوع الرسالة. على وجه التحديد، ما هي بنية البيانات التي تتضمنها الرسالة.

```
ros2 interface show geometry_msgs/msg/Twist
```

عند تنفيذ الأمر السابق ينتج:

```
# This expresses velocity in free space
broken into its linear and angular parts.
    Vector3  linear
```

```
            float64 x
            float64 y
            float64 z
    Vector3  angular
            float64 x
            float64 y
            float64 z
```

يخبرك هذا أن العقدة /turtlesim تتوقع رسالة بها متجهين، خطي وزاوي، من ثلاثة عناصر لكل منهما. إذا كنت تتذكر البيانات التي رأيناها /teleop_turtle وهي تُمرر إلى /turtlesim باستخدام أمر echo، فهي في نفس البنية:

```
linear:
  x: 2.0
  y: 0.0
  z: 0.0
angular:
  x: 0.0
  y: 0.0
  z: 0.0
  ---
```

الآن بعد أن أصبحت لديك بنية الرسالة، يمكنك نشر البيانات على موضوع مباشرةً من سطر الأوامر باستخدام:

```
ros2 topic pub <topic_name> <msg_type>
'<args>'
```

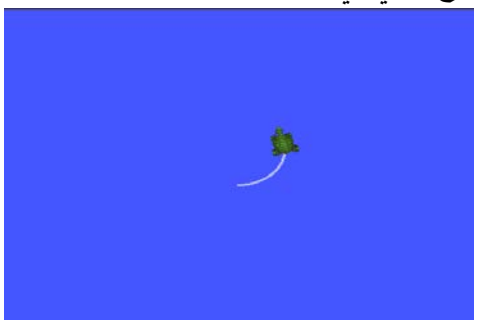
الوسيطة "<args>" هي البيانات الفعلية التي ستمررها إلى الموضوع، في البنية التي اكتشفتها للتو.

من المهم ملاحظة أن هذه argument يجب أن يتم إدخالها في بنية YAML. أدخل الأمر الكامل كما يلي:

```
ros2 topic pub --once /turtle1/cmd_vel
geometry_msgs/msg/Twist "{linear: {x: 2.0, y:
0.0, z: 0.0}, angular: {x: 0.0, y: 0.0, z:
1.8}}"
```

--once هي وسيطة اختيارية تعني "انشر رسالة واحدة ثم اخرج".

ترى الإخراج التالي في المحطة، وسترى سلحفاتك تتحرك هكذا:

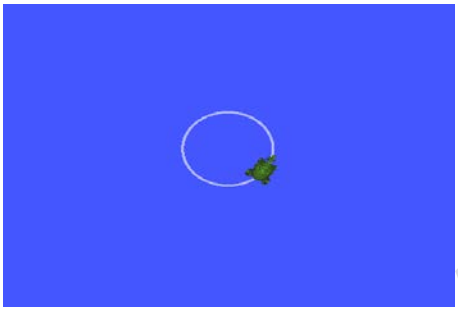


**الشكل 6 . نتيجة نشر بيانات (رسالة) عبر موضوع** /turtle1/cmd_vel

تتطلب السلحفاة (الروبوتات الحقيقية أو محاكاتها) دفقًا ثابتًا من الأوامر للعمل بشكل مستمر. لذا، لكي تستمر السلحفاة في الحركة ، يمكنك تنفيذ:

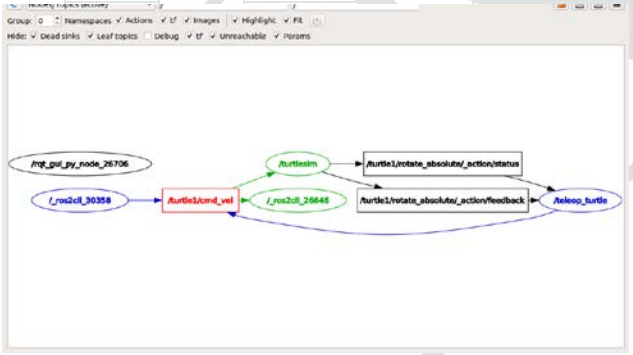
```
ros2 topic pub --rate 1 /turtle1/cmd_vel
geometry_msgs/msg/Twist "{linear: {x: 2.0, y:
0.0, z: 0.0}, angular: {x: 0.0, y: 0.0, z:
1.8}}"
```

الاختلاف هنا هو إزالة الخيار --once وإضافة الخيار --rate 1، والذي يخبر ros2 topic pub أن ينشر الأمر في دفق ثابت هو 1 هرتز.



**الشكل 7 . نتيجة تعديل الرسالة باستخدام الخيار** --rate 1

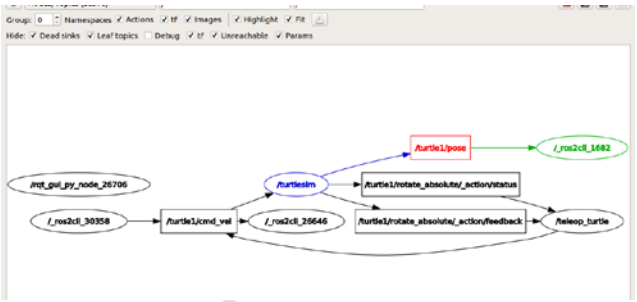
يمكنك تحديث rqt_graph لمعرفة ما يحدث بيانياً. ستلاحظ أنه عبر الأمر ros2 topic pub ... فإن العقدة (/_ros2cli_30358) تنشر فوق الموضوع /turtle1/cmd_vel، والذي يتم تلقيه بواسطة كل من العقدة ros2 topic echo ... (/_ros2cli_26646) والعقدة /turtlesim الآن.




**الشكل 8 . المخطط البياني الجديد بعد تنفيذ أمر النشر الاسبق**

أخيرًا ، يمكنك تشغيل echo على موضوع pose وإعادة فحص rqt_graph

```
ros2 topic echo /turtle1/pose
```



**الشكل 9 . تحديث المخطط بعد استخدام الأمر** echo **على** pose

يمكنك أن ترى أن العقدة /turtlesim تنشر أيضًا في موضوع pose ، الذي اشتركت فيه عقدة echo الجديدة.

للحصول على فهم أخير لهذه العملية، يمكنك عرض معدل نشر البيانات باستخدام:

```
ros2 topic hz /turtle1/pose
```

سيعيد بيانات بالمعدل الذي تنشر به العقدة /turtlesim البيانات إلى موضوع pose .

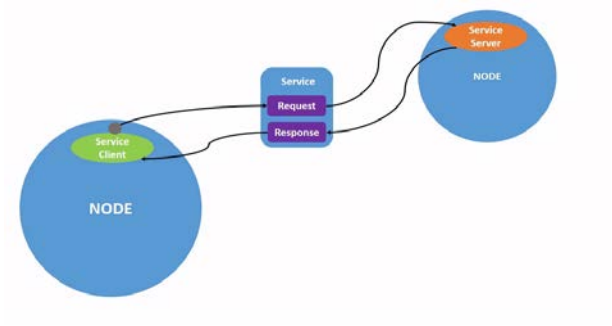
```
average rate: 59.354
  min: 0.005s max: 0.027s std dev: 0.00284s
window: 58
```

تذكر أنك قمت بتعيين معدل turtle1/cmd_vel للنشر بمعدل 1 هرتز ثابت باستخدام ros2 topic pub --rate 1 إذا قمت بتشغيل الأمر أعلاه باستخدام turtle1/cmd_vel بدلاً من turtle1/pose ، فسترى متوسطًا يعكس هذا المعدل.

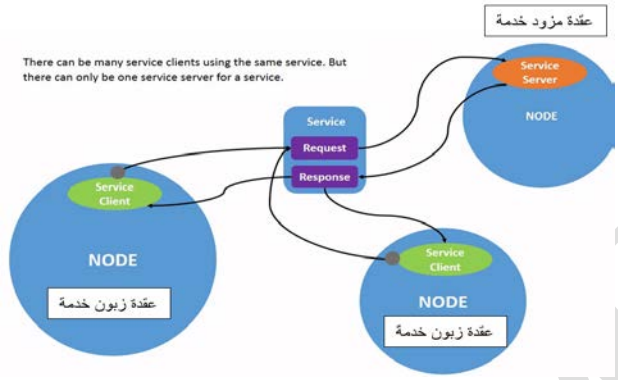
في نهاية هذه المرحلة، سيكون لديك الكثير من العقد قيد التشغيل. لا تنس إيقافهم عن طريق إدخال Ctrl + C في كل محطة terminal.

## II. فهم الخدمات SERVICES في ROS2

الخدمات Services هي طريقة أخرى للاتصال بين العقد في ROS. تعتمد الخدمات Services على نموذج الطلب والاستجابة call-and-response model، مقابل ذلك الموضوعات تعتمد نموذج بين الناشر والمشترك publisher-subscriber model. تسمح الموضوعات للعقد بالاشتراك في تدفقات البيانات والحصول على تحديثات مستمرة، بينما الخدمات تقدم البيانات فقط عندما يتم استدعاؤها على وجه التحديد من قبل العميل.




**الشكل 10 . تواصل عقدتين عبر خدمة**




**الشكل 11 . تواصل عدة عقد عبر خدمة**

افتح طرفية جديدة وقم بتشغيل:

```
ros2 run turtlesim turtlesim_node
```

افتح طرفية أخرى وقم بتشغيل:

```
ros2 run turtlesim turtle_teleop_key
```

سيؤدي تشغيل الأمر ros2 service list في new terminal إلى إرجاع قائمة بجميع الخدمات services النشطة حاليًا في النظام:

```
/clear
/kill
/reset
/spawn
/teleop_turtle/describe_parameters
/teleop_turtle/get_parameter_types
/teleop_turtle/get_parameters
/teleop_turtle/list_parameters
/teleop_turtle/set_parameters
/teleop_turtle/set_parameters_atomically
/turtle1/set_pen
/turtle1/teleport_absolute
/turtle1/teleport_relative
/turtlesim/describe_parameters
/turtlesim/get_parameter_types
/turtlesim/get_parameters
/turtlesim/list_parameters
/turtlesim/set_parameters
/turtlesim/set_parameters_atomically
```

سترى أن كلا العقدتين لهما نفس الخدمات الست مع وجود parameters في أسمائها. تحتوي كل عقدة تقريبًا في ROS 2 على infrastructure services خدمات البنية التحتية هذه التي يتم إنشاء المعلمات منها. سيكون هناك المزيد عن مفهوم parameters في الجزء التالي. بينما الآن، سيتم تجاهل parameter services من المناقشة. لاحظ الخدمات الخاصة بـ turtlesim:

/clear , /kill , /reset , /spawn , /turtle1/set_pen , /turtle1/teleport_absolute , and /turtle1/teleport_relative .

الخدمات لديها أنواع types تصف كيفية تنظيم بيانات الطلب والاستجابة لخدمة ما request and response data of a service.

يتم تعريف أنواع الخدمات Service types بشكل مشابه لأنواع الموضوعات topic types، باستثناء أنواع الخدمات التي تتكون من جزأين: رسالة للطلب وأخرى للاستجابة one message for the request and another for the response.

لمعرفة نوع الخدمة، استخدم الأمر:

```
ros2 service type <service_name>
```

ألقي نظرة على خدمة turtlesim's /clear في محطة جديدة، أدخل الأمر:

```
ros2 service type /clear
```

الذي يجب أن يعيد:

```
std_srvs/srv/Empty
```

يعني Empty type أن استدعاء الخدمة service call لا ترسل أي بيانات عند إجراء طلب making a request ولا تتلقى أي بيانات عند تلقي استجابة receiving a response.

لمعرفة أنواع جميع الخدمات النشطة في نفس الوقت، يمكنك إلحاق الخيار --show-types ، والمختصر كـ -t ، بأمر القائمة:

```
ros2 service list -t
```

الذي سيعيد:

```
/clear [std_srvs/srv/Empty]
/kill [turtlesim/srv/Kill]
/reset [std_srvs/srv/Empty]
/spawn [turtlesim/srv/Spawn]
...
/turtle1/set_pen [turtlesim/srv/SetPen]
/turtle1/teleport_absolute
[turtlesim/srv/TeleportAbsolute]
/turtle1/teleport_relative
[turtlesim/srv/TeleportRelative]
...
```

إذا كنت تريد العثور على جميع الخدمات من نوع معين، فيمكنك استخدام الأمر:

```
ros2 service find <type_name>
```

على سبيل المثال، يمكنك العثور على جميع الخدمات من نمط Empty مثل هذا:

```
ros2 service find std_srvs/srv/Empty
```

الذي سيعيد:

```
/clear
/reset
```

يمكنك استدعاء الخدمات من سطر الأوامر، ولكن عليك أولاً معرفة بنية وسائط الإدخال:

```
ros2 interface show <type_name>
```

جرب هذا على النوع Empty وهو نمط الخدمة لـ /clear :

```
ros2 interface show std_srvs/srv/Empty
```

الذي يجب أن يعيد:

```
---
```

يفصل --- بنية الطلب request (أعلاه) عن بنية response الاستجابة (أدناه). ولكن، كما علمت سابقًا، لا يرسل النوع Empty أي بيانات أو يستقبلها. لذلك، بطبيعة الحال، هيكلها فارغ.

لنفحص خدمة من النوع الذي يرسل البيانات ويستقبلها، مثل الخدمة /spawn والتي كانت من نتائج قائمة خدمة ros2 service list -t ، نعرف نوع /spawn هو turtlesim/srv/Spawn .

لمشاهدة request and response arguments لخدمة /spawn ، قم بتشغيل الأمر:

```
ros2 interface show turtlesim/srv/Spawn
```

والذي سيعيد:

```
float32 x
float32 y
float32 theta
string name # Optional.  A unique name will
be created and returned if this is empty
---
string name
```

تخبرنا المعلومات الموجودة أعلى السطر --- بالمعاملات اللازمة لاستدعاء خدمة /spawn . تحدد x و y و theta

الموضع ثنائي الأبعاد للسلحفاة المولودة، ومن الواضح أن الاسم اختياري.

المعلومات الموجودة أسفل الخط ليست شيئًا تحتاج إلى معرفته في هذه الحالة، ولكنها يمكن أن تساعدك في فهم نمط بيانات الاستجابة التي تحصل عليها من الاستدعاء.

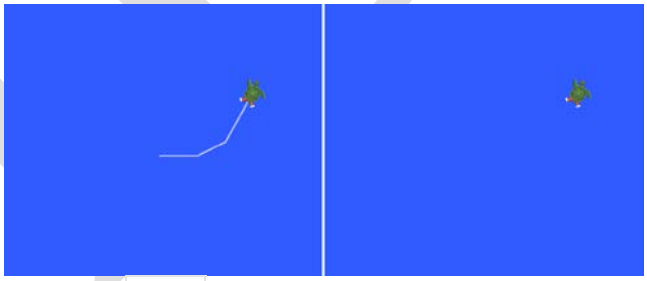
الآن بعد أن معرفة نوع الخدمة، وكيفية العثور على نوع الخدمة، وكيفية العثور على بنية وسيطات هذا النوع، يمكنك استدعاء الخدمة باستخدام:

```
ros2 service call <service_name>
<service_type> <arguments>
```

الجزء <arguments> اختياري. على سبيل المثال، أنت تعلم أن نوع الخدمات Empty ليس لها أي arguments:

```
ros2 service call /clear std_srvs/srv/Empty
```

سيؤدي هذا الأمر إلى مسح turtlesim window لأي خطوط رسمتها سلحفاتك.



**الشكل 12 . نتيجة تنفيذ الخدمة** /clear

الآن دعونا ننتج سلحفاة جديدة عن طريق استدعاء calling خدمة /spawn وتعيين arguments. إدخال <arguments> في استدعاء الخدمة من سطر الأوامر يجب أن يكون في صيغة YAML.

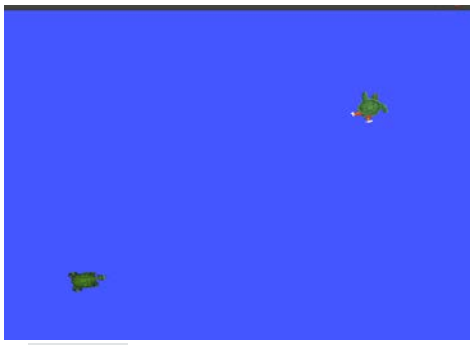
أدخل الأمر:

```
ros2 service call /spawn turtlesim/srv/Spawn
"{x: 2, y: 2, theta: 0.2, name: ''}"
```

ستحصل على عرض لما يحدث ، ثم تتم استجابة الخدمة:

```
requester: making request:
turtlesim.srv.Spawn_Request(x=2.0, y=2.0,
theta=0.2, name='')
response:
turtlesim.srv.Spawn_Response(name='turtle2')
```

سيتم تحديث نافذة turtlesim الخاصة بك مع السلحفاة التي تم تكاثرها حديثًا على الفور:



**الشكل 13 . نتيجة تنفيذ الخدمة** /spawn

## III. خلاصة

تنشر العقد معلومات عبر الموضوعات، مما يسمح لأي عدد من العقد الأخرى بالاشتراك في تلك المعلومات والوصول إليها.

يمكن للعقد التواصل باستخدام الخدمات في ROS2 على عكس الموضوع – نمط اتصال أحادي الاتجاه حيث تنشر العقدة معلومات يمكن أن يستهلكها مشترك واحد أو أكثر – الخدمة هي نمط طلب/استجابة request/response حيث يقوم العميل بتقديم طلب إلى عقدة المزودة للخدمة والخدمة تقوم بمعالجة الطلب وإنشاء استجابة. لا نستخدم عمومًا استخدام الخدمة service للمكالمات المستمرة continuous calls؛ المواضيع topics أو حتى الإجراءات actions ستكون أكثر ملاءمة.

في هذا الجزء، قمنا بفحص الروابط connections بين عدة عقد عبر مواضيع باستخدام rqt_graph وأدوات سطر الأوامر command line tools. كما استخدمنا command line tools و rqt لتحديد الخدمات واستكشافها واستدعائها. يجب أن يكون لديك الآن فكرة جيدة عن كيفية تحرك البيانات في نظام ROS2.

## المراجع:


[1]. https://www.ros.org/
[2]. http://wiki.ros.org/
[3]. http://wiki.ros.org/ROS/Tutorials