sphinx_ros Documentation

Release 0.1.1.dev19+gf28bcc7

M.J.W. Snippe

Feb 12, 2020

Contents:

CHAPTER 1

Getting started

1.1 Configuration

ros_msg_reference_version
The used ROS (Robot Operating System) version to use when referencing to default message types, e.g. `'kinetic'` or `'melodic'`.

Default

```
ros_msg_reference_version = 'melodic'
```

ros_add_package_names
Can be set to `False` to prevent package names from showing in message, service, or action type descriptions. Defaults to `True`.

Default

```
ros_add_package_names = True
```

ros_api_msg_packages
This is a list of default ROS packages that contain message or service types that are

referenced. This enables automatically linking to the ROS documentation. It should be a list of strings containing package names.

Default

```
ros_api_msg_packages = ['std_msgs', 'geometry_msgs', 'sensor_msgs']
```

New in version 0.2.

1.2 Directives

`.. ros:package::` package
Similar to the Python domain's `.. py:module::` directive. It will not output any nodes, but serves to set the context's ROS package and will produce a hyperlink target and an index entry for package. Defined packages can be referenced with `:ros:pkg:package`.

Options

- **noindex** – Prevents adding the package to the index, basically turns this directive into *ros:currentpackage*.
- **deprecated** – Flags this package as deprecated. This wil show up in the index.

`.. ros:currentpackage::` package
Similar to the Python domain's `.. py:currentmodule::` directive. It will not produce any nodes nor an index entry but will set the context's ROS package such that Sphinx knows that we are documenting stuff in that package.

This directive has no options.

`.. ros:message::` message
Can be used to describe a message type definition. It will create an index entry and a hyperlink target for this message type. It will also output nodes to describe the message.

Options

- **noindex** – Prevents adding the message to the index and creating a hyperlink target node.
- **deprecated** – Flags this message as deprecated. This wil show up in the index.

Two flags are recognized in this directive's content: `:msg_param <name>:` and `:msg_paramtype <name>:`. The former defines a parameter that is contained in the message, the latter defines the same parameter's type. All parameters will be grouped in a list.

`.. ros:service::` service
Can be used to describe a service type definition. It will create a hyperlink target for the service type. It will also output nodes to describe the service.

Options

- **noindex** – Prevents creating a hyperlink target node for the service.
- **deprecated** – Flags this service as deprecated.

Four flags are recognized in this directive's content: `:req_param <name>:` and `:req_paramtype <name>:` work similar to the flags of the message directive, but they add parameters to the service's request. `:resp_param <name>:` and `:resp_paramtype <name>:` do the same for the service's response.

.. **ros:action::** action
Can be used to describe a action type definition. It will create a hyperlink target for the action type. It will also output nodes to describe the action.

Options

- **noindex** – Prevents creating a hyperlink target node for the action.
- **deprecated** – Flags this action as deprecated.

Six flags are recognized in this directive's content: `:goal_param <name>:` and `:goal_paramtype <name>:` work similar to the flags of the message directive, but they add parameters to the action's goal. `:result_param <name>:` and `:result_paramtype <name>:` do the same for the action's result. `:feedback_param <name>:` and `:feedback_paramtype <name>:` do the same for the action's feedback.

.. **ros:node::** node
Can be used to describe a ROS node.

Options

- **noindex** – Prevents creating a hyperlink target node for the node.
- **deprecated** – Flags the node as deprecated.

Three pairs of flags are recognized in this directive's content: `:publisher <topic>: <description>` and `:publisher_msg_type <topic>: <msg_type>`, `:subscriber <topic>: <description>` and `:subscriber_msg_type <topic>: <msg_type>`, `:service <name>: <description>` and `:service_type <name>: <srv_type>`. With the first, publishers can be documented with the topics they publish on, as well as the message type that the topic expects. The second acts similar for subscribers, and with the last you can document services that the node provides, with their name and required service type.

New in version 0.2.

1.2.1 autoros

.. **ros:automessage::** message

Todo: Add description.

New in version 0.2.

.. **ros:autoservice::** service

Todo: Add description.

New in version 0.2.

.. **ros:autoaction::** action

Todo: Add description.

New in version 0.2.

.. **ros:autopackage::** package
New in version 0.2.

1.3 Roles

:ros:pkg:
Can be used to reference a defined package.

:ros:msg:
Can be used to reference a defined message type. Adding the ~ prefix to the message name will let it print *only* the message name and not the package name. First it is checked if the message is one of the ROS primitive message types (**bool**, **int8**, **uint8**, **int16**, **uint16**, **int32**, **uint32**, **int64**, **uint64**, **float32**, **float64**, **string**, **time**, **duration**). If so, it will not link anywhere. If it is of the type **Header** or it is a message in one of the default ROS message packages, it will link to the proper documentation, keeping into account the ROS version set by *ros_msg_reference_version*.

Changed in version 0.2: The default ROS message packages can be set with *ros_api_msg_packages*.

:ros:srv:
Can be used to reference a defined service type. Adding the ~ prefix to the service name will let it print *only* the service name and not the package name.

:ros:act:
Can be used to reference a defined action type. Adding the ~ prefix to the action name will let it print *only* the action name and not the package name.

:ros:node:
Can be used to reference a defined ROS node. Addding the ~ prefix to the node name will let it print *only* the node name and not the package name.

New in version 0.2.

1.4 Package example

1.4.1 The sphinx_ros_example package.

The *sphinx_ros_example* package contains all sorts of ROS objects, purely for example purposes. Objects can be referenced just like the familiar default Sphinx references, e.g. *sphinx_ros_example/Foo* will link the proper message, and *sphinx_ros_example/Bar* will link to the proper service. We can also use the ~ to prevent displaying the package name, e.g. *Foo* still points to the right message.

Author J. Doe

Maintainer J. Doe

Links

- Repository
- Bugtracker

Version 1.2

License MIT

Dependencies

Build

- `message_generation`
- `std_msgs`

Build export `std_msgs`

Build tool `catkin`

Execution

- `message_runtime`
- `std_msgs`

Messages

message sphinx_ros_example/**Foo**

Parameters

- **header** (Header) – Header of the message.
- **pose** (geometry_msgs/Pose) – The 3D pose of the foo that is detected.
- **color** (string) – The color of the foo.

Services

service sphinx_ros_example/**Bar**

Request parameters **one_way** (*sphinx_ros_example/Foo*) – The request parameter.

Response parameters

- **or_another** (int8) – The response parameter.
- **highway** (uint16) – The correct way.

Actions

action sphinx_ros_example/**FooBar**

Goal parameters **setpoint** (geometry_msgs/Point) – The setpoint to reach.

Result parameters **steady_state_error** (geometry_msgs/Point) – Error between achieved point and setpoint.

Feedback parameters

- **tracking_error** (geometry_msgs/Point) – Error between ideal trajectory and current trajectory.
- **power** (float32[]) – Current power usage per joint.

Nodes

node sphinx_ros_example/**bar_foo**

Publishers **/bar/foo** – Publishes the foo to the bar.

Subscribers **/bar/foo** – Subscribes to the bar, to listen for foo.

Services **/set_foo** (*Bar*) – Sets the foo.

Listing 1: Source

```
.. ros:package:: sphinx_ros_example

###################################
The sphinx_ros_example package.
###################################

.. contents::
  :local:
  :depth: 1

The :ros:pkg:sphinx_ros_example package contains all sorts of ROS objects,
purely for example purposes. Objects can be referenced just like the familiar
default Sphinx references, e.g. :ros:msg:sphinx_ros_example/Foo will link
the proper message, and :ros:srv:sphinx_ros_example/Bar will link to the
proper service. We can also use the ~ to prevent displaying the package
name, e.g. :ros:msg:~sphinx_ros_example/Foo still points to the right
message.

:Author: J. Doe <j.doe@mail.com>_

:Maintainer: J. Doe <j.doe@mail.com>_

:Links: * Repository <http://github.com/user/repo>_
        * Bugtracker <http://github.com/user/repo/issues>_

:Version: 1.2

:License: MIT


************
Dependencies
************

:Build: * :ros:pkg:message_generation
        * :ros:pkg:std_msgs

:Build export: :ros:pkg:std_msgs

:Build tool: :ros:pkg:catkin

:Execution: * :ros:pkg:message_runtime
            * :ros:pkg:std_msgs


********
Messages
********

.. ros:message:: Foo
```

(continues on next page)

(continued from previous page)

```
  :msg_param header: Header of the message.
  :msg_paramtype header: :ros:msg:Header
  :msg_param pose: The 3D pose of the foo that is detected.
  :msg_paramtype pose: :ros:msg:geometry_msgs/Pose
  :msg_param color: The color of the foo.
  :msg_paramtype color: :ros:msg:string


********
Services
********

.. ros:service:: Bar

  :req_param one_way: The request parameter.
  :req_paramtype one_way: :ros:msg:sphinx_ros_example/Foo
  :resp_param or_another: The response parameter.
  :resp_paramtype or_another: :ros:msg:int8
  :resp_param highway: The correct way.
  :resp_paramtype highway: :ros:msg:uint16


*******
Actions
*******

.. ros:action:: FooBar

  :goal_param setpoint: The setpoint to reach.
  :goal_paramtype setpoint: :ros:msg:geometry_msgs/Point
  :result_param steady_state_error: Error between achieved point and setpoint.
  :result_paramtype steady_state_error: :ros:msg:geometry_msgs/Point
  :feedback_param tracking_error: Error between ideal trajectory and current
                                  trajectory.
  :feedback_paramtype tracking_error: :ros:msg:geometry_msgs/Point
  :feedback_param power: Current power usage per joint.
  :feedback_paramtype power: :ros:msg:float32[]


*****
Nodes
*****

.. ros:node:: bar_foo

  :publisher /bar/foo: Publishes the foo to the bar.
  :publisher_msg_type /far/boo: :ros:msg:geometry_msgs/Point
  :subscriber /bar/foo: Subscribes to the bar, to listen for foo.
  :subscriber_msg_type /far/boo: :ros:msg:geometry_msgs/Point
  :service /set_foo: Sets the foo.
  :service_type /set_foo: :ros:srv:~sphinx_ros_example/Bar
```

1.5 Indices

This Sphinx extension adds two autogenerated indices. One for the documented packages and one for the documented message types.

With the `latex` builder, these are added automatically at the end of the document.

CHAPTER 2

Modules

2.1 sphinx_ros module

Sphinx extension adding several directives to document ROS packages.

sphinx_ros.**setup**(*app*)
Adds the ROS domain to the Sphinx application and the labels to the ROS indices to the standard domain. It also adds the configuration values *ros_add_package_names*, *ros_msg_reference_version* and *ros_api_msg_packages*.

Parameters **app** (*sphinx.application.Sphinx*) – The Sphinx application

2.2 sphinx_ros.domain module

This module defines the ROS domain. It defines three object types (messages, services, and actions) and registers the roles and directives in the Sphinx application.

class sphinx_ros.domain.**RosDomain**(*env*)
The actual domain class.

add_package(*name*, *deprecated*)
Adds a package to the domain data.

Parameters

- **name** (*str*) – The name of the package
- **deprecated** (*bool*) – Indicates whether the package is deprecated.

Returns The unique anchor of the package.

Return type str

find_obj(*env*, *pkgname*, *name*, *type*, *searchmode=0*)
Find a ROS object for name, perhaps using pkgname.

2.3 sphinx_ros.indices module

This modules defines the indices added to Sphinx.

class sphinx_ros.indices.**RosMessageIndex**(*domain*)
Index listing the documented message types.

class sphinx_ros.indices.**RosPackageIndex**(*domain*)
Index listing the documented packages.

2.4 sphinx_ros.directives module

class sphinx_ros.directives.**RosActionDirective**(*name*, *arguments*, *options*, *content*, *lineno*, *content_offset*, *block_text*, *state*, *state_machine*)
Description of a ROS action type.

class sphinx_ros.directives.**RosCurrentPackageDirective**(*name*, *arguments*, *options*, *content*, *lineno*, *content_offset*, *block_text*, *state*, *state_machine*)
This directive is just to tell Sphinx that we're documenting stuff in this package, but links to this package will not lead here.

class sphinx_ros.directives.**RosMessageDirective**(*name*, *arguments*, *options*, *content*, *lineno*, *content_offset*, *block_text*, *state*, *state_machine*)
Description of a ROS message type.

class sphinx_ros.directives.**RosNodeDirective**(*name*, *arguments*, *options*, *content*, *lineno*, *content_offset*, *block_text*, *state*, *state_machine*)
Description of a ROS node.

class sphinx_ros.directives.**RosObject**(*name*, *arguments*, *options*, *content*, *lineno*, *content_offset*, *block_text*, *state*, *state_machine*)
Description of a general ROS object.

add_object_to_domain_data(*fullname*, *obj_type*)
Add the object to the object lists of the ROS domain data.

get_index_text(*pkgname*, *name*)
Return the text for the index entry of the object.

get_object_type_prefix()
May return an optional name prefix that defines object type, e.g. 'msg' or 'srv'.

get_signature_prefix(*sig*)
Return a prefix to put before the object name in the signature.

handle_signature(*sig*, *signode*)
Transform a ROS signature into rST nodes.

Return (fully qualified name of the thing, package name if any).

If inside a package, the current package name is handled intelligently: * it is stripped from the displayed name if present * it is added to the full name (return value) if not present

class sphinx_ros.directives.**RosPackageDirective**(*name*, *arguments*, *options*, *content*, *lineno*, *content_offset*, *block_text*, *state*, *state_machine*)

Directive to mark description of a new package.

class sphinx_ros.directives.**RosServiceDirective**(*name*, *arguments*, *options*, *content*, *lineno*, *content_offset*, *block_text*, *state*, *state_machine*)

Description of a ROS service type.

class sphinx_ros.directives.**RosType**(*name*, *arguments*, *options*, *content*, *lineno*, *content_offset*, *block_text*, *state*, *state_machine*)

Super class for messages, services, and actions.

#TODO A lot of methods should be moved to RosObject, to simplify. RegEx in RosObject is not needed.

2.5 sphinx_ros.xref_role module

CHAPTER 3

Changelog

- : Added `autoros` functionality, automatically generating ROS package documentation, ROS message type documentation, ROS service type documentation, and ROS action type documentation.
- #2: Added *`ros_api_msg_packages`* configuration value.
- : Added the *`ros:node`* directive to document nodes and the *`ros:node`* role to reference them.
- #1: Fixed length arrays are also recognized.
- : Added a change log using releases.
- : Link ‘downloads’ badge to pipystats.org
- : Initial functionality
- : Added documentation on RTD (Read The Docs).

Python Module Index

S

Package Index

S

Message Type Index

f

Index

S