Inhaltsverzeichnis

This page describes the XML format used for roslaunch (/roslaunch) `.launch` files. For background on roslaunch (/roslaunch), its functionality, and related tools, please consult the roslaunch (/roslaunch) page first.

1. Evaluation order

`roslaunch` evaluates the XML file in a single pass. Includes are processed in depth-first traversal order. Tags are evaluated serially and the **last setting wins**. Thus, if there are multiple settings of a parameter, the last value specified for the parameter will be used.

Relying on the override behavior can be brittle. There is no guarantee that an override is specified correctly (e.g. if a parameter name is changed in an included file). Instead, it is recommended that override behavior be done using `$(arg)`/`<arg>` settings.

2. substitution args

Roslaunch tag attributes can make use of *substitution args*, which roslaunch will resolve prior to launching nodes. The currently supported substitution args are:

`$(env ENVIRONMENT_VARIABLE)`

Substitute the value of a variable from the current environment. The launch will fail if environment variable is not set. This value cannot be overridden by `<env>` tags.

`$(optenv ENVIRONMENT_VARIABLE)` `$(optenv ENVIRONMENT_VARIABLE default_value)`

Substitute the value of an environment variable if it is set. If `default_value` is provided, it will be used if the environment variable is not set. If `default_value` is not provided, an empty string will be used. `default_value` can be multiple words separated by spaces.

Examples:

```
  <param name="foo" value="$(optenv NUM_CPUS 1)" />
  <param name="foo" value="$(optenv CONFIG_PATH /home/marvin/ros_wor
kspace)" />
  <param name="foo" value="$(optenv VARIABLE ros rocks)" />
```

`$(find pkg)`

e.g. `$(find rospy)/manifest.xml`. Specifies a *package-relative path*. The filesystem path to the package directory will be substituted inline. Use of package-relative paths is highly encouraged as hard-coded paths inhibit the portability of the launch configuration. Forward and backwards slashes will be resolved to the local filesystem convention.

`$(anon name)`

e.g. `$(anon rviz-1)`. Generates an anonymous id based on `name`. `name` itself is a unique identifier: multiple uses of `$(anon foo)` will create the same "anonymized" name (/Names). This is used for <node> `name` attributes in order to create nodes (/Nodes) with anonymous names (/Names), as ROS requires nodes to have unique names. For example:

```
<node name="$(anon foo)" pkg="rospy_tutorials" type="talker.py" />
<node name="$(anon foo)" pkg="rospy_tutorials" type="talker.py" />
```

Will generate an error that there are two <node>s with the same name.

`$(arg foo)`

`$(arg foo)` evaluates to the value specified by an <arg> (/roslaunch/XML/arg) tag. There must be a corresponding <arg> (/roslaunch/XML/arg) tag *in the same launch file* that declares the arg.

For example:

```
<param name="foo" value="$(arg my_foo)" />
```

Will assign the *my_foo* argument to the *foo* parameter.

Another example:

```
<node name="add_two_ints_server" pkg="beginner_tutorials" type="add_
two_ints_server" />
<node name="add_two_ints_client" pkg="beginner_tutorials" type="add_
two_ints_client" args="$(arg a) $(arg b)" />
```

Will launch both the server and client from the <add_two_ints> (/ROS/Tutorials/WritingServiceClient%28c%2B%2B%29) example, passing as parameters the value *a* and *b*. The resulting launch project can be called as follows:

```
roslaunch beginner_tutorials launch_file.launch a:=1 b:=5
```

`$(eval <expression>)` **New in Kinetic**

`$(eval <expression>)` allows to evaluate arbitrary complex python expressions.

For example:

```
<param name="circumference" value="$(eval 2.* 3.1415 * arg('radiu
s'))"/>
```

Will compute the circumference from the *radius* argument and assign the result to an appropriate parameter.

Note: As a limitation, `$(eval)` expressions need to span the whole attribute string. A mixture of other substitution args with `eval` within a single string is **not** possible:

```
<param name="foo" value="$(arg foo)$(eval 6*7)bar"/>
```

To remedy this limitation, all substitution commands are available as functions *within* `eval` as well:

```
"$(eval arg('foo') + env('PATH') + 'bar' + find('pkg')"
```

For your convenience, arguments are also implicitly resolved, i.e. the following two expressions are identical:

```
"$(eval arg('foo'))"
"$(eval foo)"
```

`$(dirname)` **New in Lunar**

`$(dirname)` returns the absolute path to the directory of the launch file in which it appears. This can be used in conjunction with `eval` and `if/unless` to modify behaviour based on the installation path, or simply as a convenience for referencing `launch` or `yaml` files relative to the current file rather than relative to a package root (as with `$(find PKG)`).

For example:

```
<include file="$(dirname)/other.launch" />
```

Will expect to find `other.launch` in the same directory as the launch file which it appears in.

Substitution args are currently resolved on the local machine. In other words, environment variables and ROS package paths will be set to their values in your current environment, **even for remotely launched processes**.

3. if and unless attributes

All tags support `if` and `unless` attributes, which include or exclude a tag based on the evaluation of a value. "1" and "true" are considered true values. "0" and "false" are considered false values. Other values will error.

`if=value` (optional)

If `value` evaluates to true, include tag and its contents.

`unless=value` (optional)

Unless `value` evaluates to true (which means if `value` evaluates to false), include tag and its contents.

3.1 Example

```
<group if="$(arg foo)">
  <!-- stuff that will only be evaluated if foo is true -->
</group>

<param name="foo" value="bar" unless="$(arg foo)" />  <!-- This param wo
n't be set when "unless" condition is met -->
```

4. Tag Reference

- <launch> (/roslaunch/XML/launch)
- <node> (/roslaunch/XML/node)
- <machine> (/roslaunch/XML/machine)
- <include> (/roslaunch/XML/include)
- <remap> (/roslaunch/XML/remap)
- <env> (/roslaunch/XML/env)
- <param> (/roslaunch/XML/param)
- <rosparam> (/roslaunch/XML/rosparam)
- <group> (/roslaunch/XML/group)
- <test> (/roslaunch/XML/test)
- <arg> (/roslaunch/XML/arg)

5. Example .launch XML Config Files

NOTE: by convention, the roslaunch XML files are named with the extension `.launch`, e.g. `example.launch`.

5.1 Minimal Example

The following example shows a minimal launch configuration script. It launches a single 'talker' node, which is part of the 'rospy_tutorials' package. This node will launch on the local machine using the currently configured ROS environment (i.e. ROS_ROOT, etc...).

```
<launch>
  <node name="talker" pkg="rospy_tutorials" type="talker" />
</launch>
```

5.2 A More Complicated Example

```
<launch>
  <!-- local machine already has a definition by default.
       This tag overrides the default definition with
       specific ROS_ROOT and ROS_PACKAGE_PATH values -->
  <machine name="local_alt" address="localhost" default="true" ros-root="/u/us
er/ros/ros/" ros-package-path="/u/user/ros/ros-pkg" />
  <!-- a basic listener node -->
  <node name="listener-1" pkg="rospy_tutorials" type="listener" />
  <!-- pass args to the listener node -->
  <node name="listener-2" pkg="rospy_tutorials" type="listener" args="-foo arg
2" />
  <!-- a respawn-able listener node -->
  <node name="listener-3" pkg="rospy_tutorials" type="listener" respawn="true"
 />
  <!-- start listener node in the 'wg1' namespace -->
  <node ns="wg1" name="listener-wg1" pkg="rospy_tutorials" type="listener" res
pawn="true" />
  <!-- start a group of nodes in the 'wg2' namespace -->
  <group ns="wg2">
    <!-- remap applies to all future statements in this scope. -->
    <remap from="chatter" to="hello"/>
    <node pkg="rospy_tutorials" type="listener" name="listener" args="--test"
 respawn="true" />
    <node pkg="rospy_tutorials" type="talker" name="talker">
      <!-- set a private parameter for the node -->
      <param name="talker_1_param" value="a value" />
      <!-- nodes can have their own remap args -->
      <remap from="chatter" to="hello-1"/>
      <!-- you can set environment variables for a node -->
      <env name="ENV_EXAMPLE" value="some value" />
    </node>
  </group>
</launch>
```

5.3 Setting parameters

You can also set parameters on the Parameter Server (/Parameter%20Server). These parameters will be stored on the Parameter Server before any nodes are launched.

You can omit the `type` attribute if value is unambiguous. Supported types are `str`, `int`, `double`, `bool`. You can also specify the contents of a file instead using the `textfile` or `binfile` attributes.

Example:

```
<launch>
  <param name="somestring1" value="bar" />
  <!-- force to string instead of integer -->
  <param name="somestring2" value="10" type="str" />

  <param name="someinteger1" value="1" type="int" />
  <param name="someinteger2" value="2" />

  <param name="somefloat1" value="3.14159" type="double" />
  <param name="somefloat2" value="3.0" />

  <!-- you can set parameters in child namespaces -->
  <param name="wg/childparam" value="a child namespace parameter" />

  <!-- upload the contents of a file to the server -->
  <param name="configfile" textfile="$(find roslaunch)/example.xml" />
  <!-- upload the contents of a file as base64 binary to the server -->
  <param name="binaryfile" binfile="$(find roslaunch)/example.xml" />

</launch>
```


Wiki: roslaunch/XML (zuletzt geändert am 2017-07-21 01:19:06 durch MikePurvis (/MikePurvis))