Variable-Length Arrays as ROS Messages

Wyatt Newman

October, 2015

We have seen how to define custom messages in ROS, using a simple example including a header, an int32 and a float64. One very useful extension of the ROS message primitives is the ability to send/receive arbitrary-length vectors of message types. To illustrate this, refer to the package: cwru_msgs (within the cwru_msgs repository). The package cwru_msgs contains a "msg" folder with a message definition called "VecOfDoubles.msg". The contents of this message type is simply:

```
float64[] dbl_vec
```

When this message is received, it can be interpreted as a C++ "vector" of doubles. Two example nodes within the cwru_msgs package are written to illustrate the use of C++ vectors: cwru_msg_trivial_publisher and cwru_msg_trivial_subscriber.

Within both of the respective source-code files, we include the header:

```
#include <cwru_msgs/VecOfDoubles.h> //this is the message type we are testing
```

Then within the body of the publisher code, we create an instance of the new message type:

```
cwru_msgs::VecOfDoubles vec_msg; //create an instance of this message type
```

The variable-length array within this message can be resized manually, e.g.:

```
vec_msg.dbl_vec.resize(3); //manually resize it to hold 3 doubles
```

After setting the size, one can access elements of this array conventionally, e.g.:

```
vec_msg.dbl_vec[2]=3.1416;
```

Alternatively, one can use the vector member function "push_back()" to append data to an existing array, e.g.:

```
vec_msg.dbl_vec.push_back(counter); // this makes the vector longer, to hold additional data
```

When this message is published, a subscriber can receive it in full, even though the length is variable. In the corresponding illustrative subscriber node, the callback function uses the new message type:

```
void myCallback(const cwru_msgs::VecOfDoubles& message_holder)
```

The contents of the message can be copied to a vector type:

```
vector <double> vec_of_doubles = message_holder.dbl_vec;
```

The code can then interrogate the vec_of_doubles vector to determine its length:

```
int nvals = vec_of_doubles.size(); //ask the vector how long it is
```

This information can be used in a loop to examine the contents, which may be accessed like a normal array:

```
for (int i=0;i<nvals;i++) {
  ROS_INFO("%f",vec_of_doubles[i]); //print out all the values
  // note: with corresponding publisher, this vector gets longer each publication
}
```

When running the two illustrative nodes, the subscriber reveals that the received message is longer each time.

In practice one must be careful of two issues:

1. Make sure your variable-length vectors do not get overly large. It is all too easy to forget that the vector continues to grow, and thus you could consume all of your memory and all of your communications bandwidth as the message gets too large.
2. As with conventional arrays, if you try to access memory that has not been allocated, you will get errors (typically a segmentation fault). E.g., if your received message copied to "vec_of_doubles" is 3 elements long, trying to access vec_of_doubles[3] (the 4th element) will result in run-time errors (but no compiler warnings).